New Design and Analysis Techniques for Post-Quantum Cryptography

by

Edward Eaton

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2022

© Edward Eaton 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Dominique Unruh Professor, Institute of Computer Science, University of Tartu
Supervisor(s):	Douglas Stebila Professor, Dept. of Combinatorics & Optimization, University of Waterloo
	Alfred Menezes Professor, Dept. of Combinatorics & Optimization, University of Waterloo
Internal Member:	Ian Goldberg Professor, Cheriton School of Computer Science, University of Waterloo
Internal Member:	Mohammad Hajiabadi Professor, Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	David Jao Professor, Dept. of Combinatorics & Optimization, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis consists in part of five manuscripts written for publication. Descriptions of authorship of material are as follows:

Chapter 2 is adapted from the paper "Post-Quantum Key-Blinding for Authentication in Anonymity Networks" [71] written by myself, Douglas Stebila, and Roy Stracovsky. It was published in the proceedings of Latincrypt 2021. The research was conducted at the University of Waterloo, under the supervision of Dr. Douglas Stebila, with all authors contributing ideas and writing.

Chapter 3 is adapted from the paper "Towards Post-Quantum Updatable Public-Key Encryption via Supersingular Isogenies" [67] written by myself, David Jao, Chelsea Komlo, and Youcef Mokrani. It was published in the proceedings of Selected Areas in Cryptography 2021. The research was conducted at the University of Waterloo, under the supervision of Dr. David Jao, with all authors contributing ideas and writing. Parts of the paper for which I did not contribute fully have been removed for this thesis.

Chapter 4 is adapted from the paper "Quantum Collision-Finding in Non-Uniform Random Functions" [16] written by myself, Marko Balogh, and Fang Song. It was published in the proceedings of PQCrypto 2018. The research was conducted at Portland State University, under the supervision of Dr. Fang Song, with all authors contributing ideas and writing.

Chapter 5 is adapted from the paper "A Note on the Instantiability of the Quantum Random Oracle" [69] written by myself and Fang Song. It was published in the proceedings of PQCrypto 2020. The research was conducted at Portland State University and the University of Waterloo, under the supervision of Dr. Fang Song, with both authors contributing ideas and writing.

Chapter 6 is adapted from the paper "The 'Quantum Annoying' Property of Password-Authenticated Key Exchange Protocols" [70] written by myself and Douglas Stebila. It was published in the proceedings of PQCrypto 2021. The research was conducted at the University of Waterloo, under the supervisions of Dr. Douglas Stebila, with both authors contributing ideas and writing.

Abstract

Due to the threat of scalable quantum computation breaking existing public-key cryptography, interest in post-quantum cryptography has exploded in the past decade. There are two key aspects to the mitigation of the quantum threat. The first is to have a complete understanding of the capabilities of a quantum enabled adversary and be able to predict the impact on the security of protocols. The second is to find suitable replacements for those protocols rendered insecure. In this thesis, we develop new techniques to help address these problems, in order to better prepare for the post-quantum era.

Proofs in security models that consider quantum adversaries are notoriously more challenging compared to their classical analogues. The quantum random oracle model abstracts real world hash functions to a black box, but allows for superposition queries. This model is important as it often makes possible the reduction of the security of a protocol to the hardness of an underlying hard problem. We prove several results about the model itself. We provide upper and lower bounds on the ability of the adversary to find collisions in non-uniform functions in this model. We also compare the quantum random oracle model to the classical random oracle model and establish that a key aspect of their relationship to the standard model is unchanged. As well, we develop a way to model a new security property (dubbed quantum annoyingness) that considers the security of classical password-authenticated key exchange schemes in the presence of quantum adversaries, and prove the security of a recently standardized protocol in this model.

For the second problem, we show how established post-quantum problems can be used to build protocols beyond key establishment and signing. We look at two protocols, that of key-blinded signatures and updatable public-key encryption, which are variants of signature and key-establishment protocols. We show how these protocols can be instantiated by modifying existing post-quantum signature and key-establishment protocols. Both of these protocols were originally built heavily relying on the structure of the discrete logarithm problem. In instantiating the schemes with post-quantum assumptions, we also highlight how alternative mathematical structures can be adapted to achieve the same results. Finally, we provide proofs, implementations, and performance metrics for these instantiations.

Acknowledgements

I would firstly like to thank my supervisor Douglas Stebila for his incredible support over the past four years. His mentorship, advice, and effort were essential to every step of the development of this work, from the title to the bibliography and everything in between. I am constantly inspired by his dedication to his students, and consider myself especially lucky to count myself amongst such a privileged group.

Thank you as well to my co-supervisor Alfred Menezes, who has supported my career in academia and beyond in innumerable ways since its beginnings. It is the greatest source of pride in my life that cryptographers as brilliant as my supervisors have believed in me and invested so much in my scholarship.

Thank you to my co-authors Marko Balogh, David Jao, Chelsea Komlo, Youcef Mokrani, Fang Song, Douglas Stebila, and Roy Stracovsky. It has been a true joy to have worked with such brilliant and talented individuals on these results.

A special thanks to the members of my examination committee: Dominique Unruh, Douglas Stebila, Alfred Menezes, Ian Goldberg, Mohammad Hajiabadi, and David Jao. This work has benefited tremendously from their careful reading and feedback.

Lastly, I would like to thank my Winnipeg family: Ellen, John, and Maggie, and my Waterloo family: Ada, Darwyn, Greta, Gus, Jose, Maple, and Max. The support and love that they have given me over the years is truly what made this work possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The possible construction of quantum computers in the near future has entirely upended public key cryptography. It is not immediately obvious why quantum mechanics should lend itself so well to the computation of certain problems. The fact that being able to control the entanglement, superposition, and evolution of quantum states allows one to factor large numbers is one of the most surprising results in computer science. The original application for quantum computers was to simulate quantum systems, a natural fit. But in 1994 Peter Shor devised an algorithm that ran on a quantum computer and was capable of factoring large numbers and solving discrete logarithms quickly.

While an incredible breakthrough for the potential of quantum computation, the result also revealed a weakness in public key cryptography. All public key cryptography in use at the time (and practically to this day) is based on precisely the presumed hardness of the problems that Shor's algorithm solves. Should a quantum computer capable of running Shor's algorithm on large instances be built, it could calculate the private key associated with any public key, immediately compromising the authentication and confidentiality of essentially all electronic communication. Furthermore, encrypted data that has been stored can become decryptable, compromising past communication. Thus for any messages sent today with long-term confidentiality requirements, Shor's algorithm is a threat.

Since Shor's breakthrough, quantum technologies have moved from an academic potentiality to an industry reality. The threat of an adversary equipped with a quantum computer has become more urgent. Interest in post-quantum cryptography increased significantly after August of 2015, when the NSA announced their intention to transition their recommended cipher suite away from classical cryptography and towards quantum-resistant protocols [4]. Soon after, the National Institute of Standards and Technology (NIST) announced that they would be soliciting proposals for post-quantum key encapsulation methods (KEMs) and digital signature schemes [121]. Over the next four years, NIST has reduced the number of candidate proposals from 69 to 7 finalists: four KEMs and three signature schemes. NIST hopes to publish draft standards for some of these schemes some time in the next few years.

The timeline of post-quantum cryptography has so far been dominated by this competition and the need to get the basic building blocks of public-key cryptography 'up and running', so to speak. But classical cryptography has been used in a myriad of ways beyond these straightforward ways. With the NIST standardization process drawing to a conclusion, the time has come to focus on how to make the next generation of cryptographic primitives post-quantum.

The discrete logarithm problem is particularly flexible and strong when it comes to designing cryptographic protocols. This is due to its structure as a homomorphism, its 'tweakable' security assumption[1], and the efficiency of computations in the group. This is not the case with most post quantum primitives, which may have some of these properties, but, so far, never all. Post-quantum primitives generally require more bandwidth and have less usable structure to them. Building post-quantum alternatives to classical systems is therefore often non-trivial, if at all possible.

But the actual construction of post-quantum replacements is only the start of preparation for quantum adversaries. The other key aspect is to accurately forecast the capabilities of quantum computers. It is obvious that a quantum computer can solve the discrete logarithm problem quickly enough to necessitate the shift to post-quantum cryptography. What is less obvious is how quantum computers impact symmetric primitives. Showing the expected number of function queries needed to brute-force a search space of size N is a trivial task classically. But showing the optimality of Grover's algorithm, requiring $\sqrt{N}$ quantum function queries, is a significantly more complex task.

Furthermore, quantum computers force us to change how we model adversaries. Even if a symmetric primitive, such as a hash function, is secure against a quantum computer, an adversary could still instantiate it on a quantum computer and evaluate it in superposition. For security models like the random oracle model this challenges assumptions baked into the model. The consequences of allowing superposition evaluation for the ability to prove the security of a protocol can be unexpected, and must be thoroughly understood.

In this thesis, we will address some of the questions that arise from these challenges. We look at some classical schemes that introduce minor variations on signing and key estab-

[1]Assumptions such as the security of the one-more discrete logarithm problem [22] are variations that are still believed to be secure.

lishment protocols, and investigate the mathematical structure that enables these variants. We then look at post-quantum schemes and consider for which ones that mathematical structure is present, and for others, if the scheme can be recovered using other methods. We also develop some new results on quantum security models, proving results on the abilities of quantum adversaries relative to an oracle, showing a separation between the 'real world' and some quantum models, and considering an alternative way to model quantum adversaries for some protocols.

1.1 Constructing Post-Quantum Primitives

One of the fundamental challenges in post-quantum cryptography is the inflexibility in the underlying primitives. For classical primitives such as RSA or elliptic curve cryptography (ECC), the behaviour of the underlying mapping often has appealing properties. For example, the RSA trapdoor function is close to a permutation on the domain, a fact that is used in analyzing the security of the Full Domain Hash signature scheme [55]. Group element exponentiation is a commutative homomorphism, which is exploited to no end to construct all kinds of discrete-logarithm protocols.

For post-quantum primitives, our mappings do not behave nearly so nicely. In part, this is the nature of building cryptography that can resist quantum computers. When a problem has too much structure, a quantum computer can often step in to solve it.[2] So the fundamental challenge in post-quantum cryptography is to figure out how to 'sand down' the edges of post-quantum primitives and get them to work in the way that the classical protocols we are used to will behave.

We often categorize post-quantum cryptography into five main areas:

1. **Lattices.** Arguably the largest area of post-quantum cryptography. 'Lattices' here refers to a discrete additive subgroup, and there is an entire zoo of mathematical problems and relations between those problems that are relevant to cryptography. Most modern protocols however are based on one of three problems, or a ring variant of the problem: learning with errors (LWE), short integer solution (SIS), and the NTRU problem. In LWE, an adversary is given a matrix A and a value $t = As + e$, where e and possibly s are drawn from a special narrow distribution. Depending on

[2]This is usually because quantum computers are especially good at a class of problems called period finding. Very roughly put, the quantum Fourier transform allows us to find how long it takes a sequence to repeat itself, which unexpectedly allows us to solve all kinds of problems that have enough structure to them.

the problem statement, they then must either find s or distinguish t from a uniform element. Without the error term e, this problem is quite easy, but the inclusion of even a small error appears to make it exponentially hard for reasonable parameter sets. For SIS, given a matrix A an adversary must find a short vector s such that $As \equiv 0$. Finally, NTRU takes place over more specific rings, and requires it to be hard to find short x, y in that ring such that $hx - y = 0 \pmod{q}$ for a given h.

Due to the relative flexibility (compared to other post-quantum categories) of the operations that lattices admit, many cryptographic primitives are possible to build with lattices. Notably, this includes incredibly advanced constructions such as fully homomorphic encryption (FHE) [76, 78]. In NIST's standardization process for post-quantum cryptography, five of the seven finalists and two of the eight alternates are lattice-based, which indicates both the many ways that lattices can be used to construct secure and efficient protocols and also their relative popularity in the field. Finalists for key establishment are KYBER [130], NTRU [52], and SABER [59], while signature finalists are DILITHIUM [110] and FALCON [127]. Lattice-based schemes often boast the fastest speeds of any category, and in general have small to medium sized keys and ciphertexts/signatures.

2. **Hash / Symmetric Based.** This area of cryptography attempts to build protocols that rely solely on the one-wayness of a symmetric function F. Because of this restriction, very few protocols can be built out of such schemes. Generally only signature schemes and closely related protocols can be built out of such a simple function. Notable examples include SPHINCS+ [91], Picnic [148], and LegRoast [34]. Protocols in this category are often characterized by having small public keys, but large signatures and slow signing/verifying routines.

3. **Isogenies.** Often considered the newest area of post quantum cryptography. An isogeny is a rational mapping from one elliptic curve to another. The usage in cryptography comes from the fact that for an elliptic curve E and isogeny ϕ, it is difficult to recover ϕ from E and $\phi(E)$ alone. This as well as the fact that for some classes of curves and isogenies, the mapping can be made to be commutative has allowed many protocols to be built. Notable key exchange protocols include SIDH [98] and its modern form, SIKE [97], as well as the variant CSIDH [49]. For a long time it stood as an open problem to build an efficient signature scheme from isogenies. Recently, this problem has seen significant advancement in the form of signature schemes CSI-FiSh [38] and SQI-Sign [61]. In general, isogeny-based protocols are characterized by having small keys and signatures / ciphertexts, but being rather slow and computationally expensive.

4. **Error Correcting Codes.** This is the oldest of the post-quantum cryptography areas, dating back to 1978 [115]. In error correction, a generator matrix is used to encode information in a redundant manner, and a parity check matrix is used to decode information after errors have possibly been introduced. To turn this into a public key encryption system, a randomized form of the generator matrix can be used as a public key with the parity check matrix as the associated secret key. Encrypting a message can then be done by encoding a session key and introducing errors. Then to decrypt, the parity check matrix can be used to recover the session key. The security thus relies on the difficulty of recovering a parity check matrix from a suitably randomized generator matrix.

   Due to their relatively long history, code based systems are seen as a more conservative option, which partially makes up for the generally large public keys. Attempts to build code based signature schemes have historically not performed well. The round three finalist Classic McEliece [7] and alternates BIKE [14] and HQC [5] are all code based KEMs.

5. **Multivariate.** This subfield is based on multivariate quadratic mappings, i.e., given an input $(x_1, \ldots, x_n) \in \mathbb{F}_p^n$, the output in $\mathbb{F}_p^m$ can be described as m polynomials in the x_i's with each term having degree at most 2. In general, such mappings are one-way, i.e., hard to invert.

   The most common trick in multivariate systems is to construct the mapping so that it can be secretly decomposed into individual parts that can be easily inverted. This allows the bearer of the key to invert the mapping, allowing for a kind of full-domain hash construction. As this mapping is generally surjective, but not injective ($m < n$), the mapping is usually used to construct signature schemes, but not key establishment.

   Multivariate schemes in round 3 are the finalist Rainbow [62] and the alternate GeMSS [48]. Due to the quadratic complexity of describing the mapping, public keys are usually quite large, although the signatures are generally quite small.

Between these five core categories, researchers have developed a variety of methodologies to build both key establishment and signing protocols. These are certainly the most essential parts of public key cryptography, underpinning core protocols such as TLS, SSH, Bitcoin, and countless more. But many more protocols take standard signing or key establishment schemes and apply small modifications to better suit a specific context. These variations often take advantage of the specific properties of the discrete logarithm problem to add additional features to a standard signature or key establishment protocol.

This raises additional problems with the transition to post-quantum cryptography. The post-quantum schemes to be standardized lack the properties used in these variants. Protocols need to either be redesigned to make use only of a simple signature scheme and KEM, or find ways that the new post-quantum schemes can be similarly modified.

This forms the first main theme of this thesis. We examine two protocols based on the discrete logarithm problem that are simple tweaks on standard signature and key establishment protocols. We consider what it takes to replicate these tweaks using post-quantum protocols. This is done with a particular eye towards what mathematical properties were required by the variants, and what mathematical properties of the various post-quantum schemes can enable these variants.

Key-Blinding In Chapter 2, we consider the problem of how to add a key-blinding functionality to various post-quantum signature schemes. Key-blinding is used in Tor to transform public keys so that authentication is still possible, but the identity public key is masked.

We show how four post-quantum signature schemes can be extended to allow for key-blinding. First we consider the lattice-based scheme Dilithium-QROM [103], which is a variant of the NIST finalist Dilithium [110]. This scheme uses homomorphic properties to allow for key-blinding, similar to current solutions that extend Ed25519. Next we look at the isogeny-based scheme CSI-FiSh [38]. This variant makes use of the fact that CSI-FiSh instantiates a group action that is free and transitive, making the blinding efficient and conceptually simple.

Two more signature schemes based on generic zero-knowledge proof frameworks are considered: Picnic [148] (a NIST round 3 alternate) and LegRoast [34]. As well, a generic framework for proving the unlinkability property of key-blinding schemes is provided. Proofs of security and detailed descriptions of the protocols are provided for Dilithium-QROM, CSI-FiSh, and LegRoast. Implementations are described with performance comparisons to the versions of the signature scheme that do not support blinding.

Chapter 2 is adapted from the paper "Post-Quantum Key-Blinding for Authentication in Anonymity Networks" written by myself, Douglas Stebila, and Roy Stracovsky. It was published in the proceedings of Latincrypt 2021, and the original paper can be found on the Cryptology ePrint Archive at https://ia.cr/2021/963.

Updatable Public-Key Encryption In Chapter 3 we continue to examine variants on discrete logarithm systems, this time considering key establishment. The focus here is on the notion of ‘updatable public-key encryption’, or UPKE. UPKE has been proposed for usage in the MLS messaging protocol to help ensure the post-compromise and forward secrecy of the group key exchange protocol.

Similar to the previous chapter, we decouple and refine security definitions from an isolated use case. We then discuss how one can use techniques from CSI-FiSh, also used in the previous chapter, to build a post-quantum UPKE scheme.

This chapter is adapted from the paper “Towards Post-Quantum Updatable Public-Key Encryption via Supersingular Isogenies” written by myself, David Jao, Chelsea Komlo, and Youcef Mokrani. It was published in the proceedings of Selected Areas in Cryptography, and the original paper can be found on the Cryptology ePrint Archive at https://ia.cr/2020/1593.

1.2 Modelling Quantum Adversaries

To trust the usage of a new cryptographic protocol, we typically expect the protocol to have a proof of security. Such a proof is usually an algorithmic reduction showing that the existence of an efficient adversary who follows a certain set of rules, and yet is still able to break the security of the scheme, implies that there exists an efficient algorithm capable of breaking some underlying problem believed to be difficult. As an example, we can consider the Schnorr signature scheme. The security proof for Schnorr signatures considers the existential unforgeability property in the random oracle model. This means that the adversary (modelled as a polynomial-time black box) has access to a hashing oracle and a signing oracle and is challenged to find a signature for any message that was not submitted to the signing oracle. The proof establishes that should such an adversary exist, we can use a technique known as the forking lemma to run it multiple times in order to obtain the solution to a discrete logarithm problem. Thus an efficient (polynomial time) adversary that succeeds implies a slightly less efficient (but still polynomial time) way to find discrete logarithms.

In considering this proof (and its applicability to the real world security) there are three key attributes: the underlying problem, the proof, and the model. We want an underlying problem that is believably hard, a proof that is logically sound, and modelling that reflects a real world adversary’s abilities. Quantum computation upends all three of these attributes. Problems such as the discrete logarithm problem are no longer hard to a quantum computer, proof techniques such as the forking lemma are not logically sound,[3] and implicit assumptions made in modelling, such as queries to the random oracle being classical, do not hold when the adversary is capable of computing a hash function in superposition.

In order to address this issue, many researchers have moved their proofs to consider a fully quantum adversary. One notable example is the quantum random oracle model [40]. As the random oracle model replaces a hash function with an oracle $\mathcal{O} : x \mapsto H(x)$, the quantum random oracle model allows this oracle to be queried in superposition, i.e., performs the mapping

$$\mathcal{O} : \sum_{x,y} \alpha_{x,y} |x\rangle |y\rangle \mapsto \sum_{x,y} \alpha_{x,y} |x\rangle |y \oplus H(x)\rangle.$$

This impacts many techniques that are used in random oracle model proofs, such as monitoring the queries that an adversary makes in order to extract a witness, or programming certain inputs in a subtle way in order to force the adversary to produce a meaningful computation.

This has resulted in an area of research focused on understanding quantum computation models in a cryptographic context. A series of results has attempted to clarify questions related to what, if any, advantage quantum computation provides to an adversary. There have been many advancements in finding ways to elevate proof techniques used in the random oracle model to the quantum random oracle model [64, 68, 142, 149, 151]. In this section of the thesis, we provide a few original contributions to this problem, helping to clarify where advantages come from (or do not come) when an adversary has access to quantum resources.

Finding Collisions in Non-Uniform Functions Chapter 4 investigates the generic ability of quantum computers to find a collision in a non-uniform function. For uniform functions with a codomain with size N, it is well understood that $\Theta(N^{1/3})$ quantum queries are necessary and sufficient [150]. To characterize non-uniform functions, we focus on the min-entropy, i.e., the negative logarithm of the probability of the most likely point in the codomain. For a function with min-entropy k, we show that a quantum adversary requires at least $\Omega(2^{k/3})$ queries to find a collision with constant probability. Setting $N = 2^k$ shows that this matches the uniform case.

In addition to the generic lower-bound, we consider upper bounds, i.e., what is the performance of known quantum algorithms on distributions. We also consider upper and

[3] In the specific case of the forking lemma, this is due to the no-cloning theorem for quantum computation.

lower bounds for more specific distributions with min-entropy k: the *flat* distribution, which is uniform on a subset of size 2^k, and the δ-min-k distribution, which has a single mode with probability 2^{-k} and the rest of the codomain is uniform (with $N > 2^k$).

Chapter 4 is adapted from the paper "Quantum Collision-Finding in Non-Uniform Random Functions" written by myself, Marko Balogh, and Fang Song. It was published in the proceedings of PQCrypto 2018, and the original paper can be found on the Cryptology ePrint Archive at https://ia.cr/2017/688.

Instantiating the Quantum Random Oracle Classically, it has long been known that there exist schemes that can be proven to be secure in the random oracle model that are insecure in the standard model, when the random oracle is instantiated with an actual hash function. This implies a separation between the random oracle model and the standard model: in general, ROM proofs do not imply standard model security proofs.

In Chapter 5 we show that the same is true for the quantum random oracle model. I consider two signature schemes that establish such a separation, and in both cases show that the scheme is also secure in the QROM.

The first scheme is the original, classical result from Canetti, Goldreich, and Halevi that relies on Computationally Sound (CS) proofs. The separation technique requires the signature scheme to sign very long messages, and so the second scheme only signs short messages, which establishes that the separation holds either way.

Chapter 5 is adapted from the paper "A Note on the Instantiability of the Quantum Random Oracle" written by myself and Fang Song. It was published in the proceedings of PQCrypto 2020, and the original paper can be found on the Cryptology ePrint Archive at https://ia.cr/2019/1466.

Quantum "Annoying" Password-Authenticated Key-Exchange Chapter 6 considers what it means to be "quantum annoying". Informally, a scheme is quantum annoying if a quantum adversary can compromise its security, but only by solving a large number of discrete logarithm problems. The idea of a scheme being quantum annoying came up during a process by the Crypto Forum Research Group (CFRG) to decide on a Password-Authenticated Key-Exchange (PAKE) to be recommended for usage in Internet Engineering Task Force (IETF) protocols. It was informally observed that some of the schemes had this property, meaning that an adversary trying to compromise the security would seemingly need to solve a discrete logarithm for each guess of the password, and then only succeed when they guessed correctly. Such a scheme is thus not necessarily secure against quantum adversaries (as there is a polynomial time attack), but it can change the economic incentives for an attacker. One scheme in particular that was believed to be quantum annoying was CPace [85], which was eventually recommended for usage.

However, the property was only informally described and reasoned about. In this chapter, we discuss the question of how to formally reason about the quantum annoying property, and how a scheme can be proven to be quantum annoying. We propose a security model for the quantum annoying property, which takes the well-known Bellare, Pointcheval, Rogaway (BPR) model [23] for PAKEs, sets it in the generic group model, and provides the adversary with a (classical) discrete logarithm oracle. While this falls short of considering a fully quantum adversary, it at least establishes the additional power gained by solving discrete logarithms. In this model, we are able to show that a simplified variant of CPace is indeed quantum annoying.

Chapter 6 is adapted from the paper "The 'Quantum Annoying' Property of Password-Authenticated Key Exchange Protocols" written by myself and Douglas Stebila. It was published in the proceedings of PQCrypto 2021, and the original paper can be found on the Cryptology ePrint Archive at `https://ia.cr/2021/696`.

Chapter 2

Post-Quantum Key-Blinding

2.1 Introduction

Among the many difficulties in building a robust anonymity network, authenticating entities is a unique challenge that cannot be solved with typical techniques. Most networks will accomplish authenticity goals through the use of a signature scheme, but in a network with anonymity goals, the public keys used for signing can run contrary to those goals. One technique to overcome these conflicting goals is used in the Tor network: *key-blinding*.

A signature scheme with key-blinding works similarly to a regular signature scheme, but with the added property that given a public key pk and a nonce τ, a new public key pk_τ can be derived, which in turn can be used for signing and verification. This is useful in contexts where two parties wish to exchange signed material, but must do so in the presence of a potential eavesdropper who may attempt to de-anonymize them. Tor describes such a scheme, and its use, in version 3 of the rendezvous specification, describing how clients connect to onion services in the network [139]. In Section 2.2 we will describe precisely how key-blinding is used in Tor, and the security it is meant to provide.

It is useful to describe the key-blinding scheme as it exists in Tor, to gain some intuition for how such a scheme works and what security it provides. Key-blinding in Tor today uses the Ed25519 signature scheme [30]. Keys in this signature scheme are made with respect to a generator B of a cyclic group of size ℓ (written with additive notation). Secret keys are an integer $a \in \{1, \ldots, \ell - 1\}$ and the corresponding public key is $A = aB$. We refer to [30] for a complete description of the signing and verification processes, but for our description, it suffices to know that any such (a, A) pair are a valid key pair for Ed25519.

To blind a public key A with a nonce τ, one computes a value $t \leftarrow H(\tau || A)$, with $t \in \{1, \ldots, \ell - 1\}$. Then the blinded public key is tA, with corresponding secret key $t \cdot a \pmod{8\ell}$. This forms a new key pair that is entirely compatible with Ed25519, so that it can be used for signing and verification.

It is fairly easy to see why this scheme has the desired security properties. Given two blinded keys and the associated nonces, there is no way to tell if they come from the same identity public key or not. Without knowledge of the identity public key, the distribution of the blinded public key is entirely uniform over the public key space, so that these keys are entirely unlinkable to each other. Furthermore, the keys retain their unforgeability, as (informally put) the blinded secret key ta requires both t and a to be known. Formal proofs of the security properties can be found in a tech report posted to the Tor developer mailing list [90].

This system works quite well for Tor today, but with the development of quantum computers, cryptography based on the discrete logarithm problem will eventually be rendered insecure. To ensure the long-term security of Tor, a replacement post-quantum signature scheme with key-blinding will be needed.

Chapter Contributions and Structure. In this chapter we address the challenge of extending post-quantum signature schemes to have a key-blinding functionality. We consider four promising post-quantum signature schemes. *Dilithium* is a lattice-based signature scheme that is currently under consideration in NIST's Post-Quantum Cryptography standardization effort [65]. Instead of directly working with Dilithium, we will work with the Dilithium-QROM variant [103]. Dilithium-QROM has simpler provable guarantees by neatly fitting into the 'Lossy ID scheme' framework [2], so we work within the same framework to ensure that our scheme has similar guarantees. *CSI-FiSh* is a relatively new post-quantum signature scheme based on the CSIDH group action [38]. With both Dilithium and CSI-FiSh we are able to establish that they have enough mathematical structure to enable key-blinding. For Dilithium we exploit a homomorphism between the secret and public key spaces, and only require a small change to the signing and verification procedures. For CSI-FiSh we use the fact that the scheme instantiates a free and transitive group action, which provides enough structure to add on key-blinding with essentially no change to the signing procedure.

Picnic is another submission to NIST's efforts, which constructs a signature scheme out of the 'MPC-in-the-head' paradigm [93, 50], proving knowledge of a secret encryption key associated with a ciphertext. *LegRoast* is based on the Picnic framework, but replaces the more traditional symmetric function used with the Legendre PRF, the homomorphic

Scheme	$\lvert pk \rvert$	$\lvert \sigma \rvert$	KeyGen	Blind	Sign	Verify
Dilithium-QROM	7.7 kB	5.7 kB	3810 ms	-	9360 ms	2890 ms
blDlithium-QROM	10 kB	5.7 kB	2180 ms	1650 ms	28300 ms	717 ms
Increase from blinding	1.3×	1×	0.6×	-	3×	0.25×
LegRoast	0.50 kB	7.94 kB	0.9 ms	-	12.4 ms	11.7 ms
blLegRoast	0.50 kB	11.22 kB	0.9 ms	0.9 ms	18.6 ms	17.8 ms
Increase from blinding	1.0×	1.4×	1.0×	-	1.5×	1.5×
CSI-FiSh-Merkleized	32 B	1.8–2.1 kB	10900 ms	-	559 ms	559 ms
CSI-FiSh-unMerkleized	16 kB	0.45 kB	10800 ms	-	554 ms	553 ms
blCSI-FiSh	16 kB	0.45 kB	10600 ms	10600 ms	546 ms	540 ms
Increase from blinding	1.0×	1.0×	1.0×	-	1.0×	1.0×

Table 2.1: Performance results from the implemented key-blinding schemes.

properties of which allow for small signatures [58, 34]. Key-blinding in LegRoast and Picnic does not use the same homomorphism between the keyspaces, but instead exploits the generic nature of the zero-knowledge proof underlying the protocols.

In all the schemes, blinding is generally around as efficient as key generation, while signing is either as efficient, or at worst a quarter as fast. We provide a generic framework for proving the unlinkability property, showing that it can be reduced to two straightforward properties of signature schemes. We prove all these schemes both unlinkable and unforgeable in the random oracle model. Note that each of the signature schemes we have discussed are built out of the Fiat–Shamir paradigm. We discuss why this is the case, and what some of the challenges are for building a key-blinded scheme out of a trapdoor signature scheme. Finally, we provide prototype implementations of the blinded CSI-FiSh, LegRoast, and Dilithium-QROM schemes and discuss aspects of their performance as it applies to Tor. Our results from the three implemented schemes are shown in Table 2.1. Note that we emphasize the increase over the raw numbers for the timing information. Implementations are not optimized and may not reflect how long a 'proper' implementation will take. Nonetheless, the increase reflects how much additional work is required to use the scheme for key-blinding. For all schemes, blinded public keys have the same size as their unblinded version and so we do not distinguish between the two.

To begin in Section 2.2, we provide background information on Tor and key-blinding to motivate the construction and provide context. We then provide definitions of key-blinding and its security properties for a formal framework we use for remainder of the paper in Section 2.3. Section 2.4 discusses the security property of unlinkability and establish a useful framework for proving the property.

After this set-up, we are able to dive into the details of the schemes themselves and their associated proofs. We refer to the version of each scheme that supports blinding with the prefix ‘bl’. First in Section 2.5 we describe an extension of Dilithium-QROM and in Section 2.6 an extension of CSI-FiSh. Then in Section 2.7 we show how LegRoast can be extended using somewhat different techniques, before finally sketching an outline on how Picnic can be extended in Section 2.8. For each of blDilithium-QROM, blCSI-FiSh, and blLegRoast, we provide a complete description of the scheme, specific parameters, and a detailed proof of unforgeability and unlinkability. In Section 2.9 we also discuss the results of prototype implementations of each of these schemes, comparing their performance to an unblinded version. Finally, we conclude our findings in Section 2.10.

Related Work As mentioned, the schemes that we choose to base blinded signature schemes off of are Dilithium [65, 103], CSI-FiSh [38], LegRoast / PorcRoast [34], and Picnic [50]. While to our knowledge, this is the first attempt to construct post-quantum key-blinded signatures, there are a few other papers who have attempted to build similar primitives, for different reasons. In a 2018 preprint [18], Barreto, Ricardini, Simplicio Jr., and Patil considered post-quantum PKIs in vehicle-to-anything (V2X) communications. One of the techniques they developed to provide anonymity to vehicles in such a context involved transformations on public key materials similar to that of key-blinding. Their construction was based on the lattice scheme qTESLA, which was a candidate for standardization in the first two rounds of NIST’s process. The process of key-blinding also bears a similarity to hierarchical deterministic wallets used for Bitcoin [84, 109]. These wallets allow a user to create child public and secret keys for the delegation of abilities for spending. Such a protocol has much stronger requirements than simple key-blinding, which does not need to be hierarchical and does not need for the child secret keys to contain no information about parent secret keys. Some work on post-quantum deterministic wallets has been published [8], with their scheme also based on qTESLA.

It is important to distinguish between the key-blinding schemes we discuss here and the notion of ‘blind signatures’, for which post-quantum schemes already exist [88, 124]. Blind signatures are an interactive protocol that allow a user to obtain a signature on a message without the signer knowing the message. This is very different from key-blinded schemes, which have the same functionality as a traditional signature scheme, but with the extra ability to randomize public keys.

2.2 Onion Services

The Tor network serves millions of clients a day, providing anonymity to users from the websites they connect to, and concealing what they are connecting to from their Internet service provider and any other intermediary in their path [138]. An important part of the Tor networks is *onion services* (previously known as hidden services). Onion services allow users to not only *access* content with Tor's strong privacy guarantees, but also *serve* content.

At a high level, onion services work by uploading a three hop path (called a *circuit* in Tor terminology) to a Tor node called a *introduction point*. This path begins at the introduction point and ends at the onion service. Because of Tor's layered encryption, the introduction point does not know where the onion service lives, only where the next node in the path lives. For a client to connect to the onion service, they use the `.onion` address to find the introduction point, who will then direct their communication towards the onion service.

To help clients access introduction points, the onion service uploads an encrypted list of the introduction points to distinguished nodes in the network called HSDirs (hidden service directories). The client's overall process for accessing an onion service is to (i) query the HSDir for the descriptor, which provides a list of introduction points, and then (ii) ask one of the introduction points to pass along a *rendezvous point* (another random node in the network) to the onion service, and then (iii) communicate with the onion service through the rendezvous point.

Of particular interest to us is the process by which the client obtains the descriptor from the HSDir. Because this is the first point of contact between the client and the HSDir, the client holds no information about the HSDir except for the `.onion` address (which was communicated in some out-of-band way) and the current state of the network. HSDirs are not trusted authorities in Tor, as nodes in Tor are run by volunteers, and anyone can become an HSDir. After joining the network, they will be flagged as an HSDir and included in the list of directories after some time (approximately 2–3 days), to ensure that churn does not affect the availability of services too much.

This makes the descriptor lookup challenging from a privacy perspective. HSDirs are somewhat untrusted, but clients possess no private information other than the `.onion` address. The previous version of the rendezvous specification (v2) had the `.onion` address be the hash of the onion service's public key. When a client connected to the HSDir, they would provide the `.onion` address directly, and the HSDir would respond with the full public key and the descriptor. For version 3 of the specification, a goal was to provide less information to the HSDir. For this, the technique of key-blinding is employed.

In the most recent version of the rendezvous specification (the specification that describes the process of connecting to an onion service), the `.onion` address is the long-lived EdDSA public key of the onion service. Time in the Tor network is divided into periods, with the period length a consensus parameter and the period number the number of periods that have occurred since the Unix epoch. So given a public key, a nonce, and consensus parameters of the Tor network, the *blinding factor* t is computed by hashing together the public key, the nonce, and the current period number, as well as some parameters of both the Tor network and the signature scheme. As mentioned in the introduction, this value t is treated as an integer in the range 1 to $\ell - 1$, with ℓ the order of the cyclic group, so that public keys are transformed by simply multiplying by t.

The blinded key can then be used to index the descriptors while they are held by the HSDir. Clients can derive the blinded key from the `.onion` address and query for a descriptor by providing the blinded key. So, the blinded key serves as a private *index* from which the descriptor may be queried. This also implicitly means that the client is implicitly checking the connection between the identity public key from the `.onion` address and the blinded public key. For security it is important that only the actual owner of the `.onion` address can upload a descriptor to a given index. This is where the signing functionality of key-blinding is used. Onion services also upload a signature on the descriptor, which can be verified with the blinded key. When HSDirs verify this signature, they ensure that the descriptor is being uploaded by the actual owner of the identity public key—all without knowing what the `.onion` address is.

Under the current system with Ed25519, a malicious actor with a quantum computer could forge a signature with respect to a chosen blinded public key, and use this to upload false information about an introduction point. This would mean that queries to the onion service could be redirected to the adversary.

2.3 Key-Blinding Signature Scheme Definitions

Definition 1. A key-blinding signature scheme Δ consists of four algorithms: KeyGen, $\mathsf{BlindPk}$, Sign, and Verify, where

- $\mathsf{KeyGen}()$ generates an identity key pair (pk, sk).
- $\mathsf{BlindPk}(pk, \tau)$ deterministically generates a blinded public key pk_τ.

- $\mathsf{Sign}(msg, sk, \tau)$ generates a signature σ for the message msg using the identity secret key sk and epoch τ.

- $\mathsf{Verify}(msg, \sigma, pk_\tau)$ accepts if the signature is valid under the message msg and epoch τ used to generate pk_τ, otherwise it rejects.

We require the usual correctness properties for signature schemes, but extended for key-blinding. That is, if (pk, sk) is a keypair generated from KeyGen, pk_τ is then derived from $\mathsf{BlindPk}$ with a given nonce τ, and $\sigma \leftarrow \mathsf{Sign}(msg, sk, \tau)$, then with overwhelming probability $\mathsf{Verify}(msg, \sigma, pk_\tau)$ will accept. Anyone without knowledge of the identity public key can verify using the pk_τ given in the descriptor, while someone with knowledge of the identity key can take the additional step of checking $pk_\tau = \mathsf{BlindPk}(pk, \tau)$. Note that we do not require that blinded keys can be blinded again.

Signatures with key-blinding must satisfy two security requirements. First, they must be *unlinkable*, which means that an adversary without knowledge of the identity public key who observes many public key blindings as well as signatures under those blindings cannot distinguish a fresh blinding of the public key from an entirely unrelated key. Second, the scheme must satisfy *unforgeability*. This property is largely the same for signature schemes with key-blinding as it is for typical signature schemes. However, rather than just devising an (msg, σ) such that $\mathsf{Verify}(msg, \sigma, pk)$ accepts, the adversary must be able to provide an (msg, σ, τ) such that $\mathsf{Verify}(msg, \sigma, pk_\tau)$ accepts where $pk_\tau \leftarrow \mathsf{BlindPk}(pk, \tau)$.

Earlier versions of both of these formulations appear in [90]. The security definitions that we present here are more general. The definitions in [90] were tied to the exact usage of key-blinding in Tor, and do not consider security in situations where the blinding process is decoupled from the signing process, so that multiple signatures can be issued under the same blinded public key.

Definition 2 (Unlinkability under Chosen Message and Epoch Attack)**.** Let $\Delta = (\mathsf{KeyGen}, \mathsf{BlindPk}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme with key-blinding. Then for an adversary $\mathcal{A}$ we define the 'Unlinkability under Chosen Message and Epoch Attack' experiment as follows: First, obtain $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}()$ as a fresh identity key pair. The adversary does not get access to the identity keys. The adversary does have access to the following oracles:

- A blinding oracle, which takes as input a blinding nonce τ (representing a chosen epoch) and provides $\mathsf{BlindPk}(pk_0, \tau)$ to $\mathcal{A}$.

- A signing oracle, which takes a message msg and a blinding nonce τ that has previously been queried to the blinding oracle and provides $\mathsf{Sign}(msg, sk_0, \tau)$.

Eventually the adversary may submit a challenge query τ^* that has not previously been queried to the blinding oracle. A new key pair $(pk_1, sk_1) \leftarrow \mathsf{KeyGen}()$ and a bit $b \leftarrow\!\!\$\ \{0,1\}$ is uniformly sampled. The adversary receives $pk_b^* \leftarrow \mathsf{BlindPk}(pk_b, \tau^*)$.

The adversary may continue to query the blinding and signing oracle, with the restriction that if τ^* is queried to either oracle then the keypair (pk_b, sk_b) is used. After having made a total of q_B queries to the public key blinding oracle and q_S queries to the signing oracle, the adversary submits a guess b^* for the bit b, and is said to have won if $b^* = b$.

For a signature scheme with key-blinding, let $\mathcal{A}$ be the adversary in the unlinkability experiment. Then we define their advantage as

$$\mathbf{Adv}_{\Delta}^{\mathsf{UL\text{-}CMEA}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}\right|.$$

Next, we define the notion of unforgeability. Our model for unforgeability is essentially the standard definition of existential unforgeability under chosen message attack, with the additional consideration that the adversary can make signing queries with respect to an epoch nonce τ of their choice, and has access to the identity public and secret key.

Definition 3 (Existential Unforgeability under Chosen Message and Epoch Attack)**.** Let $\Delta = (\mathsf{KeyGen}, \mathsf{BlindPk}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme with key-blinding. Then for an adversary $\mathcal{A}$ we define the 'unforgeability under chosen message and epoch attack' experiment as follows: First, obtain $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}()$ as a fresh identity key pair. The adversary is provided with pk_0. The adversary has access to an oracle that signs a chosen message with respect to a chosen epoch nonce. Specifically, they may query (msg, τ) to the signing oracle which generates $\sigma_{msg,\tau} \leftarrow \mathsf{Sign}(msg, sk, \tau)$ and sends that to the adversary.

After having made q_S queries to the signing oracle, the adversary submits a forgery $(msg^*, \sigma^*, \tau^*)$, and is said to have won if $\mathsf{Verify}(msg^*, \sigma^*, \mathsf{BlindPk}(pk_0, \tau^*))$ accepts, and (msg^*, τ^*) was not a query made to the signing oracle.

For a signature scheme with key-blinding, let $\mathcal{A}$ be the adversary in the unforgeability experiment. Then we define their advantage as

$$\mathbf{Adv}_{\Delta}^{\mathsf{EU\text{-}CMEA}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \text{ wins}]\right|.$$

2.4 Unlinkability of Signatures with Key-Blinding

We want to establish that an adversary who has access to a blinding oracle and a signing oracle still cannot distinguish a new blinding of the identity public key from the blinding of a fresh public key. We observe a common technique that could be used for showing unlinkability among the signature schemes we consider. The general framework we employ is to (i) show that the distribution of blinded public keys is independent from the value of the identity public key (over the randomness in the random oracle) and then (ii) show that the distribution of signatures is dependent *only* on the value of the blinded public key. Taken together, this means that the values provided to the adversary from the blinding and signing oracles are independent from the identity public and secret key, and thus do not leak any information.

While our techniques provide a generic framework to establish unlinkability, they do not extend to showing unforgeability or provide a way to generically *construct* schemes with key-blinding out of Fiat–Shamir style signature schemes. This is because the mechanism by which blinding is accomplished changes depending on the scheme. As a result, there is no common framework for constructing a key-blinding scheme, and the proof of unforgeability similarly must take the blinding mechanism into account.

We call the first property *independent blinding*. Informally, it states that the distribution of blinded public keys is independent from the identity public key. This means that seeing any number of blindings of a public key leaks no information on the identity public key.

We show how the second property can be established by a property we call *signing with oracle reprogramming*, which states that if we have the ability to reprogram the random oracle used in the signature scheme, then we can create signatures indistinguishable from real ones for any message. Many signature schemes show their security by first establishing just such a property. As an example, for signature schemes built from an identification protocol and the Fiat–Shamir heuristic, the zero-knowledge property is typically proven by establishing the ability to simulate transcripts given only the public key. When given control of the random oracle, we can sample transcripts and reprogram the random oracle to generate a signature.

To formalize this notion, we require a concept we call a *reprogrammed point extractor*. This is a simple function, efficiently computable and publicly known to all, which, given a signature σ, public key, and message, can extract the point on which the random oracle is reprogrammed to make the signature verify.

It is best to illustrate this with an example. Consider a generic form of the Probabilistic Signature Scheme [26] defined with respect to a trapdoor permutation T. To sign a message, sample a random salt r and compute $x = T^{-1}(H(pk\|msg\|r))$. The signature is $\sigma = (x, r)$. To verify a signature, simply check that $T(x) = H(pk\|msg\|r)$. It is straightforward to show that signatures can be generated if the random oracle can be reprogrammed. On input of a message m, sample a random (x, r) and reprogram H so that $H(pk\|msg\|r) = T(x)$. If T is a permutation, it is easy to see that (x, r) will have the same distribution as in a real signature, and the reprogramming cannot be detected as long as r is sufficiently long (so that the adversary is unlikely to have queried $pk\|msg\|r$ beforehand). Let Ext denote our reprogrammed point extractor. For the example above, we have $\mathsf{Ext}((x, r), pk, msg) = pk\|msg\|r$. Note that not all of the signature is necessarily used as part of the reprogrammed point, which is why x does not appear in it.

Definition 4 (Signing with oracle reprogramming)**.** Let Σ be a signature scheme that relies on a random oracle H. We say that the signature scheme admits signing with oracle reprogramming if there exists a reprogrammed point extractor Ext and a forgery function Forge that takes in pk, msg and returns (y, σ) such that $\Sigma.\mathsf{Verify}^{H:\mathsf{Ext}(\sigma,pk,msg)\mapsto y}(pk, msg, \sigma) \to$ 'accept', where $H : x \mapsto y$ denotes the random oracle reprogrammed such that $H(x) = y$.

In order to use oracle reprogramming to sign a message, we need to consider the probability that an adversary is capable of noticing that the real signing algorithm wasn't used. This amounts to considering the joint distribution of the signature as well as the input and output of the hash function on the reprogrammed point.

Definition 5 (Statistical distance of forgeries)**.** Let $\Sigma = (\mathsf{Sign}, \mathsf{Verify})$ be a signature scheme defined with respect to a random oracle H and a public key space $\mathcal{PK}$ that admits signing with oracle reprogramming via a point extractor Ext and a forgery process Forge.

For a public key and message pk, msg, we consider the adversary's ability to distinguish the distribution of $(y_{forged}, \sigma_{forged}) \leftarrow \mathsf{Forge}(pk, msg)$ from the distribution of $(y_{real}, \sigma_{real})$ where $\sigma_{real} \leftarrow \mathsf{Sign}(pk, msg)$ and $y_{real} = H(\mathsf{Ext}(\sigma_{real}, pk, msg))$ (i.e., the output of the hash on the input that would be reprogrammed).

We denote $L1$ distance between these distributions as δ, that is

$$\delta = \sum_{\sigma,y} \left|\Pr[\sigma_{real} = \sigma, y_{real} = y] - \Pr[\sigma_{forged} = \sigma, y_{forged} = y]\right| .$$

As well, we need to consider the ability of an adversary to detect that reprogramming has occurred. This can be evaluated by considering the min-entropy of the point that is reprogrammed, to ensure that the probability that an adversary queries this point prior to reprogramming is low.

Definition 6 (Min-entropy of extracted point)**.** Let h_{min} denote the min-entropy of $\mathsf{Ext}(\sigma, pk, msg)$, where $(y, \sigma) \leftarrow \mathsf{Forge}(pk, msg)$. Here the entropy is over the randomness in the Forge process.

Note that in the above definition we are implicitly assuming that the statistical distance and the entropy are not dependent on msg, pk, or H. For all of the schemes that we construct this is the case. Even if these values were dependent on pk, msg, or H, the scheme could still be secure as long as they were sufficiently small on average. However to simplify the proof and notation, our definition only considers schemes where they do not depend on pk, msg, or H.

We now consider the unlinkability experiment in Definition 2. We will show a reduction from an adversary who makes queries to the signing oracle to an adversary who makes none.

Lemma 1. *Let Δ be a key-blinding signature scheme which admits signing with oracle reprogramming with L1 distance δ and min-entropy of reprogrammed points h_{min}. Let $\mathcal{A}$ be an adversary making q_B queries to the blinding oracle, q_S queries to the signing oracle, and q_H queries to the (classical) random oracle. Using $\mathcal{A}$, we construct an adversary $\mathcal{A}^{\mathsf{KO}}$ that makes no signing queries (i.e., a key-only adversary) for which*

$$\mathbf{Adv}^{\Delta}_{\mathsf{UL\text{-}CMEA}}(\mathcal{A}) \leq \mathbf{Adv}^{\Delta}_{\mathsf{UL\text{-}CMEA}}(\mathcal{A}^{\mathsf{KO}}) + q_H q_S 2^{-h_{min}} + q_S \delta$$

Proof. To construct the adversary $\mathcal{A}^{\mathsf{KO}}$ while relying on the adversary $\mathcal{A}$ as a subroutine, we must show how to handle queries to the blinding oracle and the signing oracle. For queries to the blinding oracle, $\mathcal{A}^{\mathsf{KO}}$ can simply pass along these queries to the blinding oracle provided to them.

To handle the signing queries, we rely on signing with oracle reprogramming. Whenever a signing query is made with respect to a blinded public key pk_{τ_i}, we can use signing with oracle reprogramming (Definition 4) to create a signature for the adversary. We generate $(y, \sigma) \leftarrow \mathsf{Forge}(pk_{\tau_i}, msg)$ and reprogram the random oracle so that $H(\mathsf{Ext}(\sigma, pk_{\tau_i}, msg) \rightarrow y$.

By Definition 4 the resulting signature verifies, so we need to consider the adversary's ability to distinguish that the secret key is not being used to sign messages. To realize this, the adversary either needs to observe that the oracle has been reprogrammed, or notice a difference in the observed distribution of some part of the signature.

To distinguish that reprogramming has occurred during signing, the adversary must have queried the random oracle on the reprogrammed point previously. In total, q_S points will be reprogrammed. So the adversary makes q_H guesses, and then q_S points are chosen to be reprogrammed from a distribution with min entropy h_{min}, and we want to consider the probability of a match between the q_H and q_S points. We can upper-bound this by $q_H q_S 2^{-h_{min}}$.

Next we consider the output distribution of the programmed points. There are q_S reprogrammed points, and the statistical ($L1$) distance between the forged values and the real values is δ, so the adversary's advantage in distinguishing based on the distribution of reprogrammed values is at most $q_S\delta$. $\square$

We now only need to consider the advantage of $\mathcal{A}^{\mathsf{KO}}$, an adversary who makes no queries to the signing oracle. So, we need only consider how the blinding oracle and random oracle provide information to the adversary.

To characterize the security of blinding, we want to insist that the distribution of the public key returned by $\mathsf{BlindPk}$ is independent of the identity public key input, so that no knowledge is gained. However care must be taken here, because the $\mathsf{BlindPk}$ algorithm is actually deterministic on the inputs pk and τ. So when we refer to the 'distribution' of $\mathsf{BlindPk}$ we need to be clear over what randomness.

In practice, the $\mathsf{BlindPk}$ function hashes the public key and the nonce τ to generate some randomness, and then uses that randomness to blind the public key. To separate out the process of hashing to generate randomness and using the randomness, we will define a new function $\mathsf{RandBlind}(pk; r)$, which takes in a public key and some randomness, and blinds the public key. Then $\mathsf{BlindPk}$ is defined by making $\mathsf{RandBlind}$ deterministic through the random oracle H. Specifically, $\mathsf{BlindPk}(pk, \tau) = \mathsf{RandBlind}(pk; H(pk\|\tau))$.

Definition 7 (Independent Blinding)**.** Let Δ be a key-blinding signature scheme and let n be a positive integer. Let $pk_0, pk_1, \ldots, pk_n$ be public keys generated from KeyGen. Sample uniform randomness $r_1, r_2, \ldots, r_n$. The *independent blinding advantage*, denoted $\mathbf{Adv}^{\mathsf{Ind-Blind}}_{\Delta,n}(\mathcal{A})$, is the advantage that an adversary has in distinguishing the following two distributions:

1) $\mathsf{RandBlind}(pk_0; r_1), \mathsf{RandBlind}(pk_0; r_2), \ldots, \mathsf{RandBlind}(pk_0; r_n)$
2) $\mathsf{RandBlind}(pk_1; r_1), \mathsf{RandBlind}(pk_2; r_2), \ldots, \mathsf{RandBlind}(pk_n; r_n)$

This ensures that the adversary $\mathcal{A}^{\mathsf{KO}}$ may observe many blindings of the public key with respect to arbitrary nonces but what they see is close to a distribution independent of the identity public key.

Lemma 2. *Let Δ be a key-blinding signature scheme and let h_{pk} be the min-entropy of the public key returned from $\Delta.\mathsf{KeyGen}$. Let $\mathcal{A}^{\mathsf{KO}}$ be an $\mathsf{UL-CMEA}$ adversary that makes no queries to its signing oracle. Then there exists an algorithm $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathsf{UL-CMEA}}_{\Delta}(\mathcal{A}^{\mathsf{KO}}) \leq 2\mathbf{Adv}^{\mathsf{Ind-Blind}}_{\Delta,n}(\mathcal{B}) + q_H 2^{-h_{pk}},$$

where n is the number of blinding queries $\mathcal{A}^{\mathsf{KO}}$ makes to its public key-blinding oracle, q_H is the number of queries to the hash oracle, and the runtime of $\mathcal{B}$ is approximately the same as the runtime of $\mathcal{A}$.

Proof. We use a simple game-hopping proof to bound the adversary's success probability. Game G_0 proceeds according to $\mathbf{Exp}^{\mathsf{UL-CMEA}}$ with the adversary making no signing queries by assumption. In game G_1, when the adversary queries the blinding oracle with input τ, rather than responding with $\mathsf{BlindPk}(pk, \tau) = \mathsf{RandBlind}(pk, H(pk\|\tau))$, we sample a uniformly random r and return $\mathsf{RandBlind}(pk, r)$. Note that there is no difference between these games until an adversary queries $H(pk\|\tau)$ for some τ; we let bad be the event that the adversary makes such a query. Games G_0 and G_1 are identical-until-bad [27].

In game G_2 we modify the response to each blinding query from $\mathsf{RandBlind}(pk, r)$ by sampling a fresh pk' each time from KeyGen and returning $\mathsf{RandBlind}(pk', r)$. We can construct, from an adversary that distinguishes G_1 from G_2, a reduction $\mathcal{B}$ that distinguishes the two distributions in the independent blinding property: G_1 uses the first distribution in 7, whereas G_2 uses the second. Thus G_2 can be distinguished from G_1 with advantage at most $\mathbf{Adv}^{\mathsf{Ind-Blind}}_{\Delta,n}(\mathcal{B})$.

We now consider the probability of event bad—i.e., the adversary querying $H(pk\|\tau)$—in G_2. Since none of the blindings actually use pk, the success probability is bounded by the adversary's ability to guess the public key. For this we use the min-entropy of the public key returned from key-generation. Over q_H queries, the probability that an adversary is able to guess the public key is bounded by $q_H 2^{-h_{pk}}$. By the fundamental lemma of game playing [27], we can thus bound the probability that an adversary distinguishes G_0 and G_1 by this quantity and the difference between G_1 and G_2, i.e., $q_H 2^{-h_{pk}} + \mathbf{Adv}^{\mathsf{Ind-Blind}}_{\Delta,n}(\mathcal{B})$.

Finally, in game G_2 all blinded public keys are independent of the original key, so everything the adversary sees is independent of the challenge bit b, and thus the adversary's advantage in G_2 is 0, yielding the desired result. □

One could go to the effort of computing or bounding the min-entropy h_{pk} of the public key returned from KeyGen for each scheme. It is convenient to observe that for any correct signature scheme there exists a very simple adversary $\mathcal{A}$ for which $2^{-h_{pk}} \leq \mathbf{Adv}^{\mathsf{EUF-CMEA}}_{\Delta}(\mathcal{A})$. Otherwise, if a scheme has certain public keys that have an abnormally high chance of being generated, $\mathcal{A}$ can break unforgeability by running KeyGen and hoping that the desired public key (with the corresponding secret key) is generated. Thus,

Corollary 1. *Let Δ and $\mathcal{A}^{\mathsf{KO}}$ be as in Lemma 2. Then there exist algorithms $\mathcal{B}_1, \mathcal{B}_2$ such that*

$$\mathbf{Adv}^{\mathsf{UL-CMEA}}_{\Delta}(\mathcal{A}^{\mathsf{KO}}) \leq 2\mathbf{Adv}^{\mathsf{Ind-Blind}}_{\Delta,n}(\mathcal{B}_1) + q_H \mathbf{Adv}^{\mathsf{EUF-CMEA}}_{\Delta}(\mathcal{B}_2),$$

where n is the number of blinding queries $\mathcal{A}^{\mathsf{KO}}$ makes to its public key-blinding oracle and the runtimes of $\mathcal{B}_1$ and $\mathcal{B}_2$ are approximately the same as that of $\mathcal{A}$.

2.5 A Lattice-Based Key-Blinding Scheme

Dilithium [65] is a finalist in the NIST post-quantum signature standardization process and comes from a long line of lattice-based signature schemes. We present a key-blinded version of Dilithium-QROM [103] which modifies Dilithium to permit lossy key generation, hence allowing a reduction from the scheme to the Module Learning with Errors (MLWE) assumption. Later, in Section 2.5.5, we discuss the challenges in blinding Dilithium itself.

Our construction, blDilithium-QROM utilizes the fact that addition is homomorphic. As a result, the $\mathbf{A}$ matrix is a public matrix used by all parties in the network. In addition, both signing and verification use the public key when sampling the challenge c. Finally, the identity public key consists of an extra bit as this permits key-blinding.

2.5.1 blDilithium-QROM Preliminaries

Throughout this section, vectors are written in lowercase bold-face and matrices are written in uppercase bold-face. Let q be prime, then $\mathbb{Z}_q$ denotes the integers modulo q, and R and R_q denote the rings $\mathbb{Z}[x]/\langle x^n+1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ respectively.

For some $w \in \mathbb{Z}_q$, $\|w\|_\infty$ denotes $|w'|$ where $w' \in \mathbb{Z}$ such that $w' \equiv w \pmod{q}$ and $-\frac{q-1}{2} \leq w' \leq \frac{w-1}{2}$. This norm can be extended as follows. For some $w = w_{n-1}x^{n-1} + \cdots + w_1 x + w_0 \in R_q$, $\|w\|_\infty = \max\left(\|w_1\|_\infty, \ldots, \|w_{n-1}\|_\infty\right)$, and for some $\mathbf{w} = (w_1, \ldots, w_k) \in R_q^k$, $\|\mathbf{w}\|_\infty = \max\left(\|w_1\|_\infty, \ldots, \|w_k\|_\infty\right)$. We let S_η denote the set of $w \in R_q$ such that $\|w\|_\infty \leq \eta$. In addition, we also define the L2 norm for $w \in R$ as $\|w\| = \sqrt{\|w_0\|_\infty^2 + \cdots + \|w_{n-1}\|_\infty^2}$.

ChSet denotes the challenge space. In the context of identification protocols, ChSet is the set of possible challenges a verifier can submit to the prover. In the context of signature schemes arrived at via the Fiat–Shamir transform, ChSet is the output domain of the hash function H used to digest the message.

G is also a hash function that takes in elements of $R_q^k \times R_q^k \times \{0,1\}^*$ and returns elements of $S_\eta^\ell \times S_\eta^k$.

We make use of several supporting algorithms with full descriptions in [103] which extract or compute on higher and lower order bits of elements in $\mathbb{Z}_q$. These algorithms are extended to elements of R_q and R_q^k by coefficient-wise and element-wise application. We give a general description of the supporting algorithms:

- $\mathsf{Power2Round}_q(r, d)$ extracts the higher $(\log(r) - d)$ order bits of r.
- $\mathsf{HighBits}_q(r, \alpha)$ extracts the higher $(\approx \log(r) - \log \alpha)$ order bits of r.
- $\mathsf{LowBits}_q(r, \alpha)$ extracts the lower $(\approx \log \alpha)$ order bits of r.
- $\mathsf{MakeHint}_q(z, r, \alpha)$ constructs a hint to allow the computation of the higher order bits of $r + z$ without the need to store z.
- $\mathsf{UseHint}_q(h, r, \alpha)$ uses the hint to compute the higher order bits of $r + z$.

The interactions of these algorithms are outlined by Lemmas 4.1 and 4.2 of [103], which we make use of in the proof of unforgeability.

We now discuss the MLWE assumption which was introduced in [108] as a generalization of the LWE assumption introduced in [128]. We leverage the decision version of the assumption, which posits that the following problem is hard given appropriate parameter selection.

Definition 8. The decisional $\mathsf{MLWE}_{m,k,\chi}$ problem over the ring R_q is to distinguish the pair $(\mathbf{A}, \mathbf{t})$ for $\mathbf{A} \leftarrow_\$ R_q^{m\times k}, \mathbf{t} \leftarrow_\$ R_q^m$ from the pair $(\mathbf{A}, \mathbf{A}\mathbf{s}_1 + \mathbf{s_2})$ where $\mathbf{A} \leftarrow_\$ R_q^{m\times k}, \mathbf{s}_1 \leftarrow_\$ \chi(R_q^k)$, $\mathbf{s}_2 \leftarrow_\$ \chi(R_q^m)$. The $\mathsf{MLWE}_{m,k,\chi}$ advantage is defined as

$$\mathbf{Adv}^{\mathsf{MLWE}}_{m,k,\chi}(\mathcal{A}) = \left|\Pr[\mathcal{A} \text{ wins decisional } \mathsf{MLWE}_{m,k,\chi} \text{ over } R_q] - \frac{1}{2}\right|$$

We introduce a modified version of the MLWE assumption in which the $\mathbf{A}$ matrix is a parameter of decisional MLWE problem. This modified assumption is required because each signer in the anonymity network shares the same $\mathbf{A}$ matrix. We refer to this assumption as the MLWE assumption with static $\mathbf{A}$, which assumes that the following problem is hard given appropriate parameter selection.

Definition 9. The decisional $\mathsf{SA\text{-}MLWE}_{m,k,\chi,\mathbf{A}}$ problem over the ring R_q is to distinguish $\mathbf{t}$ for $\mathbf{t} \leftarrow_\$ R_q^m$ from $\mathbf{A}\mathbf{s}_1 + \mathbf{s_2}$ where $\mathbf{s}_1 \leftarrow_\$ \chi(R_q^k), \mathbf{s}_2 \leftarrow_\$ \chi(R_q^m)$. The $\mathsf{SA\text{-}MLWE}_{m,k,\chi}$ advantage is defined as

$$\mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{m,k,\chi,\mathbf{A}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \text{ wins decisional } \mathsf{SA\text{-}MLWE}_{m,k,\chi,\mathbf{A}} \text{ over } R_q] - \frac{1}{2}\right|$$

There are a few implications to using a constant (rather than per-user) $\mathbf{A}$ matrix. The first is the question of how to generate such a matrix. It is well known that there exist methods to construct an $\mathbf{A}$ matrix that is 'backdoored', so that an adversary can solve problems with respect to $\mathbf{A}$ otherwise meant to be infeasible [77]. Especially in the context of Tor, users of a constant $\mathbf{A}$ matrix need assurance that whoever generated it did not include a backdoor. One possible way to accomplish this is to generate $\mathbf{A}$ pseudorandomly from a "nothing up my sleeve" seed. With a simple and publicly available seed, hopefully users would be convinced that there is no backdoor.

The other issue is the potential for all for the price of one attacks. With everyone using the same matrix, if someone performs a large computation on $\mathbf{A}$, they might be able to determine a kind of backdoor, even if $\mathbf{A}$ was not generated with one. This causes the potential security consequences of a break to be more severe. Of course, parameters are already chosen for LWE instances so that no one should be able to break even a single instance of the problem, but such consequences should still be considered before deployment of such a system.

2.5.2 blDilithium-QROM Specification

Signing is performed by using G to sample blinding secrets adding these to the identity public key secrets, and performing the operations in KeyGen. Then, the procedure in [103] is followed except that the public key is added to the hash to produce the challenge c (this is done to prevent attacks similar to that of related-key attacks [117]), and an extra bit is included as part of the public key $\mathbf{t}_1$, so that rounding errors are not introduced when we blind the public key. We want $\mathbf{t}_0$ to be unchanged, so we remove the last bit of $\mathbf{t}_1$ first. Verification is also similar to that in [103] except that $c\mathbf{t}_{1,\tau}$ is multiplied by 2^{d-1}. Key Generation, blinding, signing, and verifying are described in full in Figure 2.1. The parameters of the scheme are identical to the parameters in [103], except that $d = 7$ and $\beta = 644$. We list them in full in Table 2.2.

In Figure 2.2 we also detail a version of blDilithium-QROM in which both $\mathbf{t}_1$ and $\mathbf{t}_0$ are both published in the identity public key. Differences between the two versions are highlighted in red. This is in keeping with the unforgeability proof in [103], and is necessary for both the proofs of unforgeability and unlinkability. Observe that signatures generated via the procedure in Figure 2.1 are identical to those generated in Figure 2.2 given the same inputs and randomness, hence the scheme in Figure 2.1 must be at least as secure as the scheme in Figure 2.2 as it simply releases less information in the identity public key.

$\text{blDilithium-QROM.KeyGen}()$

```
1 : $K \leftarrow \{0,1\}^{256}$
2 : $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow\!\$\; S_\eta^\ell \times S_\eta^k$
3 : $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
4 : $\mathbf{t}_1 \leftarrow \mathsf{Power2Round}_q(\mathbf{t}, d-1)$
5 : $\mathbf{t}_0 \leftarrow \mathbf{t} - \lfloor \mathbf{t}_1/2 \rfloor \cdot 2^d$
6 : $pk \leftarrow \mathbf{t}_1$
7 : $sk \leftarrow (\mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0, K)$
8 : return $(pk, sk)$
```

$\text{blDilithium-QROM.BlindPk}(pk = \mathbf{t}_1, \tau)$

```
1 : $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2 : $\mathbf{t}' \leftarrow \mathbf{A}\mathbf{s}'_1 + \mathbf{s}'_2$
3 : $\mathbf{t}'_1 \leftarrow \mathsf{Power2Round}_q(\mathbf{t}', d-1)$
4 : $\mathbf{t}_{1,\tau} \leftarrow \mathbf{t}_1 + \mathbf{t}'_1$
5 : $pk_\tau \leftarrow \mathbf{t}_{1,\tau}$
6 : return $pk_\tau$
```

$\text{blDilithium-QROM.Sign}(M, pk = \mathbf{t}_1, sk = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0, K), \tau)$

```
1 : $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2 : $\mathbf{s}_{1,\tau} \leftarrow \mathbf{s}_1 + \mathbf{s}'_1$
3 : $\mathbf{s}_{2,\tau} \leftarrow \mathbf{s}_2 + \mathbf{s}'_2$
4 : $\mathbf{t}_\tau \leftarrow \mathbf{A}\mathbf{s}_{1,\tau} + \mathbf{s}_{2,\tau}$
5 : $\mathbf{t}_{1,\tau} \leftarrow \mathsf{Power2Round}_q(\mathbf{t}_\tau, d-1)$
6 : $\mathbf{t}_{0,\tau} \leftarrow \mathbf{t}_\tau - \lfloor \mathbf{t}_{1,\tau}/2 \rfloor \cdot 2^d$
7 : $\kappa \leftarrow 0$
8 : while $(\mathbf{z}, \mathbf{h}) = (\perp, \perp)$ and $\kappa \leq 200/(1-\delta)$ do
9 :   $\kappa \leftarrow \kappa + 1$
10 :   $\mathbf{y} \leftarrow\!\$\; S_{\gamma'-1}^\ell$
11 :   $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y}$
12 :   $\mathbf{w}_1 \leftarrow \mathsf{HighBits}_q(\mathbf{w}, 2\gamma)$
13 :   $c \leftarrow \mathsf{H}(M \| \mathbf{w}_1 \| \mathbf{t}_{1,\tau})$
14 :   $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_{1,\tau}$
15 :   if $\|\mathbf{z}\|_\infty \geq \gamma' - \beta$ or $\left\|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_{2,\tau}, 2\gamma)\right\|_\infty \geq \gamma - \beta$ then $(\mathbf{z}, \mathbf{h}) \leftarrow (\perp, \perp)$
16 :   else $\mathbf{h} \leftarrow \mathsf{MakeHint}_q(-c\mathbf{t}_{0,\tau}, \mathbf{w} - c\mathbf{s}_{2,\tau} + c\mathbf{t}_{0,\tau}, 2\gamma)$
17 : return $\sigma = (\mathbf{z}, \mathbf{h}, c)$
```

$\text{blDilithium-QROM.Verify}(M, \sigma = (\mathbf{z}, \mathbf{h}, c), pk_\tau = \mathbf{t}_{1,\tau})$

```
1 : $\mathbf{w}'_1 \leftarrow \mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_{1,\tau} \cdot 2^{d-1}, 2\gamma)$
2 : if $\|\mathbf{z}\|_\infty < \gamma' - \beta$ and $c = \mathsf{H}(M \| \mathbf{w}_1 \| \mathbf{t}_{1,\tau})$ then return accept
3 : else return reject
```

Figure 2.1: blDilithium-QROM signature scheme.

	$\mathsf{blDilithium\text{-}QROM}$ recommended	$\mathsf{Dilithium\text{-}QROM}$ recommended
q	$2^{45} - 21283$	$2^{45} - 21283$
n	512	512
(k, ℓ)	(4,4)	(4,4)
d	7	15
$\mathsf{ChSet} = \{c \in R \mid \|c\|_\infty = 1 \land \|c\| = \dots\}$	$\sqrt{46}$	$\sqrt{46}$
γ	905679	905679
γ'	905679	905679
η	7	7
β	644	322
BKZ block-size to break LWE	480	
Best known classical bit-cost	136	
Best known quantum bit-cost	127	

Table 2.2: Parameters for the blDilithium-QROM scheme, compared with Dilithium-QROM parameters.

```
blDilithium-QROM'.KeyGen()
1:  $K \leftarrow \{0,1\}^{256}$
2:  $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow\!\$\ S_\eta^\ell \times S_\eta^k$
3:  $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
4:  $\mathbf{t}_1 \leftarrow \mathsf{Power2Round}_q(\mathbf{t}, d)$
5:  $\mathbf{t}_0 \leftarrow \mathbf{t} - \mathbf{t}_1 \cdot 2^d$
6:  $pk \leftarrow (\mathbf{t}_1, \mathbf{t}_0)$
7:  $sk \leftarrow (\mathbf{s}_1, \mathbf{s}_2, K)$
8:  return $(pk, sk)$
```

```
blDilithium-QROM'.BlindPk($pk = (\mathbf{t}_1, \mathbf{t}_0), \tau$)
1:  $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2:  $\mathbf{t}' \leftarrow \mathbf{A}\mathbf{s}'_1 + \mathbf{s}'_2$
3:  $\mathbf{t} \leftarrow \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$
4:  $\mathbf{t}_\tau \leftarrow \mathbf{t} + \mathbf{t}'$
5:  $\mathbf{t}_{1,\tau} \leftarrow \mathsf{Power2Round}_q(\mathbf{t}_\tau, d)$
6:  $\mathbf{t}_{0,\tau} \leftarrow \mathbf{t}_\tau - \mathbf{t}_{1,\tau} \cdot 2^d$
7:  $pk_\tau \leftarrow (\mathbf{t}_{1,\tau}, \mathbf{t}_{0,\tau})$
8:  return $pk_\tau$
```

```
blDilithium-QROM'.Sign($M, pk = (\mathbf{t}_1, \mathbf{t}_0), sk = (\mathbf{s}_1, \mathbf{s}_2, K), \tau$)
1:  $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2:  $\mathbf{s}_{1,\tau} \leftarrow \mathbf{s}_1 + \mathbf{s}'_1$
3:  $\mathbf{s}_{2,\tau} \leftarrow \mathbf{s}_2 + \mathbf{s}'_2$
4:  $\mathbf{t}_\tau \leftarrow \mathbf{A}\mathbf{s}_{1,\tau} + \mathbf{s}_{2,\tau}$
5:  $\mathbf{t}_{1,\tau} \leftarrow \mathsf{Power2Round}_q(\mathbf{t}_\tau, d)$
6:  $\mathbf{t}_{0,\tau} \leftarrow \mathbf{t}_\tau - \mathbf{t}_{1,\tau} \cdot 2^d$
7:  $\kappa \leftarrow 0$
8:  while $(\mathbf{z}, \mathbf{h}) = (\perp, \perp)$ and $\kappa \le 200/(1-\delta)$ do
9:      $\kappa \leftarrow \kappa + 1$
10:     $\mathbf{y} \leftarrow\!\$\ S_{\gamma'-1}^\ell$
11:     $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y}$
12:     $\mathbf{w}_1 \leftarrow \mathsf{HighBits}_q(\mathbf{w}, 2\gamma)$
13:     $c \leftarrow \mathsf{H}(M \| \mathbf{w}_1 \| pk_\tau)$
14:     $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_{1,\tau}$
15:     if $\|\mathbf{z}\|_\infty \ge \gamma' - \beta$ or $\left\|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_{2,\tau}, 2\gamma)\right\|_\infty \ge \gamma - \beta$ then $(\mathbf{z}, \mathbf{h}) \leftarrow (\perp, \perp)$
16:     else $\mathbf{h} \leftarrow \mathsf{MakeHint}_q(-c\mathbf{t}_{0,\tau}, \mathbf{w} - c\mathbf{s}_{2,\tau} + c\mathbf{t}_{0,\tau}, 2\gamma)$
17: return $\sigma = (\mathbf{z}, \mathbf{h}, c)$
```

```
blDilithium-QROM'.Verify($M, \sigma = (\mathbf{z}, \mathbf{h}, c), pk_\tau = (\mathbf{t}_{1,\tau}, \mathbf{t}_{0,\tau})$)
1:  $\mathbf{w}'_1 \leftarrow \mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_{1,\tau} \cdot 2^d, 2\gamma)$
2:  if $\|\mathbf{z}\|_\infty < \gamma' - \beta$ and $c = \mathsf{H}(M \| \mathbf{w}_1 \| pk_\tau)$ then return accept
3:  else return reject
```

Figure 2.2: blDilithium-QROM′, with extra information in the identity public key.

2.5.3 blDilithium-QROM Unforgeability

We begin by proving the unforgeability of blDilithium-QROM′ in the context of key-blinded signature schemes (see Definition 3), which we achieve by emulating the proof found in [103] while introducing the blinding procedure into relevant algorithms. At a high level, we begin by defining an identification protocol blDilithium-QROM-ID, addressing four key properties (naHVZK, correctness, lossiness, and min entropy), and applying the Fiat–Shamir transform to arrive at a signature scheme equivalent to blDilithium-QROM′. This allows us to leverage Theorem 3.1 of [103] (with the appropriate modifications to take into account blinding) to bound $\mathbf{Adv}^{\mathsf{EUF-CMEA}}_{\mathsf{blDilithium\text{-}QROM}}(\mathcal{A})$.

Non Abort Honest Verifier Zero-Knowledge

We begin by showing that blDilithium-QROM-ID is naHVZK with $\varepsilon_{\mathsf{zk}} = 0$ as defined in Definition 2.5 of [103]. This entails showing that transcripts of honest interactions of blDilithium-QROM-ID are statistically indistinguishable from the output of some transcript simulator that only has access to the public key.

Lemma 3. *If* $\max_{s \in S_\eta, c \in \mathsf{ChSet}} \|2cs\|_\infty \leq \beta$ *then* blDilithium-QROM-ID *is* naHVZK *with* $\varepsilon_{\mathsf{zk}} = 0$.

Proof. Suppose $\mathbf{s}_1, \mathbf{s}_2$ come from a valid identity keypair i.e. $\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 = \mathbf{t}$. For a given $\mathbf{z} \in S^\ell_{\gamma'-\beta-1}$ and $c \in \mathsf{ChSet}$, the probability $\mathbf{z}$ was generated in Trans is equal to

$$\Pr[\mathbf{y} \leftarrow_{\$} S^\ell_{\gamma'-1} \mid \mathbf{y} + c(\mathbf{s}_1 + \mathbf{s}'_1) = \mathbf{z}] = \Pr[\mathbf{y} \leftarrow_{\$} S^\ell_{\gamma'-1} \mid \mathbf{y} = \mathbf{z} - c(\mathbf{s}_1 + \mathbf{s}'_1)]$$

Since $\left\|c(\mathbf{s}_1 + \mathbf{s}_2)\right\|_\infty \leq \beta$, then $\mathbf{z} - c(\mathbf{s}_1 + \mathbf{s}'_1) \in S^\ell_{\gamma'-1}$ thus

$$\Pr[\mathbf{y} \leftarrow_{\$} S^\ell_{\gamma'-1} \mid \mathbf{y} = \mathbf{z} - c(\mathbf{s}_1 + \mathbf{s}_2)] = \frac{1}{\left|S^\ell_{\gamma'-1}\right|}.$$

Hence, every $\mathbf{z} \in S^\ell_{\gamma'-\beta-1}$ has an equal probability of being generated. In addition, it follows that the probability of not producing a $\mathbf{z} \in S^\ell_{\gamma'-\beta-1}$ in Trans is $1 - \left|S^\ell_{\gamma'-\beta-1}\right| / \left|S^\ell_{\gamma'-1}\right|$, so the distribution of $(c, \mathbf{z})$ is identical between Trans and Sim.

Finally, observe that

$$\mathbf{w} - c(\mathbf{s}_2 + \mathbf{s}'_2) = \mathbf{A}\mathbf{y} - c(\mathbf{s}_2 + \mathbf{s}'_2) = \mathbf{A}(\mathbf{z} - c(\mathbf{s}_1 + \mathbf{s}'_1)) - (\mathbf{s}_2 + \mathbf{s}'_2) = \mathbf{A}\mathbf{z} - c(\mathbf{t} + \mathbf{t}').$$

Thus Trans and Sim produce $\mathbf{h}$ from identical distribution as well.

As the output distributions of Trans and Sim are exactly identical, $\varepsilon_{\mathsf{zk}} = 0$. □

```
blDilithium-QROM-ID.KeyGen()
1: $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow\$ S_\eta^\ell \times S_\eta^k$
2: $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
3: $\mathbf{t}_1 \leftarrow \mathsf{Power2Round}_q(\mathbf{t}, d)$
4: $\mathbf{t}_0 \leftarrow \mathbf{t} - \mathbf{t}_1 \cdot 2^d$
5: $pk \leftarrow (\mathbf{t}_1, \mathbf{t}_0)$
6: $sk \leftarrow (\mathbf{s}_1, \mathbf{s}_2, pk)$
7: return $(pk, sk)$
```

```
blDilithium-QROM-ID.Prv_1$(sk = (\mathbf{s}_1, \mathbf{s}_2, pk), \tau)$
1: $\mathbf{y} \leftarrow\$ S_{\gamma'-1}^\ell$
2: $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y}$
3: $\mathbf{w}_1 \leftarrow \mathsf{HighBits}_q(\mathbf{w}, 2\gamma)$
4: return $(W = \mathbf{w}_1, St = (\mathbf{w}, \mathbf{y}))$
```

```
blDilithium-QROM-ID.Prv_2$(sk = (\mathbf{s}_1, \mathbf{s}_2, pk), W = \mathbf{w}_1, c, St = (\mathbf{w}, \mathbf{y}), \tau)$
1: $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2: $\mathbf{s}_{1,\tau} \leftarrow \mathbf{s}_1 + \mathbf{s}'_1$
3: $\mathbf{s}_{2,\tau} \leftarrow \mathbf{s}_2 + \mathbf{s}'_2$
4: $\mathbf{t}_\tau \leftarrow \mathbf{A}\mathbf{s}_{1,\tau} + \mathbf{s}_{2,\tau}$
5: $\mathbf{t}_{1,\tau} \leftarrow \mathsf{Power2Round}_q(\mathbf{t}_\tau, d)$
6: $\mathbf{t}_{0,\tau} \leftarrow \mathbf{t}_\tau - \mathbf{t}_{1,\tau} \cdot 2^d$
7: $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_{1,\tau}$
8: if $\|\mathbf{z}\|_\infty \geq \gamma' - \beta$ or $\|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_{2,\tau}, 2\gamma)\|_\infty \geq \gamma - \beta$ then $(\mathbf{z}, \mathbf{h}) \leftarrow (\perp, \perp)$
9: else $\mathbf{h} \leftarrow \mathsf{MakeHint}_q(-c\mathbf{t}_{0,\tau}, \mathbf{w} - c\mathbf{s}_{2,\tau} + c\mathbf{t}_{0,\tau}, 2\gamma)$
10: return $Z = (\mathbf{z}, \mathbf{h})$
```

```
blDilithium-QROM-ID.Verify$(pk = (\mathbf{t}_1, \mathbf{t}_0), W = \mathbf{w}_1, c, Z = (\mathbf{z}, \mathbf{h}), \tau)$
1: $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2: $\mathbf{t}' \leftarrow \mathbf{A}\mathbf{s}'_1 + \mathbf{s}'_2$
3: $\mathbf{t} \leftarrow \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$
4: $\mathbf{t}_\tau \leftarrow \mathbf{t} + \mathbf{t}'$
5: $\mathbf{t}_{1,\tau} \leftarrow \mathsf{Power2Round}_q(\mathbf{t}_\tau, d)$
6: $\mathbf{t}_{0,\tau} \leftarrow \mathbf{t}_\tau - \mathbf{t}_{1,\tau} \cdot 2^d$
7: if $\|\mathbf{z}\|_\infty < \gamma' - \beta$ and $\mathbf{w}_1 = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_{1,\tau} \cdot 2^d, 2\gamma)$ then return accept
8: else return reject
```

Figure 2.3: Key generation, proving, and verification algorithms for blDilithium-QROM-ID.

Correctness

We now consider the correctness error of blDilithium-QROM-ID in the sense of Definition 2.3 of [103], which involves the probability of both the prover failing and the verifier failing on a valid output of the prover.

Trans

```
1 : $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2 : $\mathbf{s}_{1,\tau} \leftarrow \mathbf{s}_1 + \mathbf{s}'_1$
3 : $\mathbf{s}_{2,\tau} \leftarrow \mathbf{s}_2 + \mathbf{s}'_2$
4 : $\mathbf{t}_\tau \leftarrow \mathbf{A}\mathbf{s}_{1,\tau} + \mathbf{s}_{2,\tau}$
5 : $\mathbf{t}_{1,\tau} \leftarrow \mathsf{Power2Round}_q(\mathbf{t}_\tau, d)$
6 : $\mathbf{t}_{0,\tau} \leftarrow \mathbf{t}_\tau - \mathbf{t}_{1,\tau} \cdot 2^d$
7 : $\mathbf{y} \leftarrow\!\!\$\; S^\ell_{\gamma'-1}$
8 : $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y}$
9 : $\mathbf{w}_1 \leftarrow \mathsf{HighBits}_q(\mathbf{w}, 2\gamma)$
10 : $c \leftarrow\!\!\$\; \mathsf{ChSet}$
11 : $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_{1,\tau}$
12 : if $\|\mathbf{z}\|_\infty \geq \gamma' - \beta$ then return $(\perp, (\perp, \perp))$
13 : if $\|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_{2,\tau}, 2\gamma)\|_\infty \geq \gamma - \beta$ then
14 :     return $(\perp, (\perp, \perp))$
15 : $\mathbf{h} \leftarrow \mathsf{MakeHint}_q(-c\mathbf{t}_{0,\tau}, \mathbf{w} - c\mathbf{s}_{2,\tau} + c\mathbf{t}_{0,\tau}, 2\gamma)$
16 : return $(c, (\mathbf{z}, \mathbf{h}))$
```

Sim

```
1 : $(\mathbf{s}'_1, \mathbf{s}'_2) \leftarrow \mathsf{G}(pk \| \tau)$
2 : $\mathbf{t}' \leftarrow \mathbf{A}\mathbf{s}'_1 + \mathbf{s}'_2$
3 : $\mathbf{t} \leftarrow \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$
4 : $\mathbf{t}_\tau \leftarrow \mathbf{t} + \mathbf{t}'$
5 : $\mathbf{t}_{1,\tau} \leftarrow \mathsf{Power2Round}_q(\mathbf{t}_\tau, d)$
6 : $\mathbf{t}_{0,\tau} \leftarrow \mathbf{t}_\tau - \mathbf{t}_{1,\tau} \cdot 2^d$
7 : with probability $1 - \frac{\left|S^\ell_{\gamma'-\beta-1}\right|}{\left|S^\ell_{\gamma'-1}\right|}$ return $(\perp, (\perp, \perp))$
8 : $\mathbf{z} \leftarrow\!\!\$\; S^\ell_{\gamma'-\beta-1}$
9 : $c \leftarrow\!\!\$\; \mathsf{ChSet}$
10 : if $\|\mathsf{LowBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}_\tau, 2\gamma)\|_\infty \geq \gamma - \beta$ then
11 :     return $(\perp, (\perp, \perp))$
12 : $\mathbf{h} \leftarrow \mathsf{MakeHint}_q(-c\mathbf{t}_{0,\tau}, \mathbf{A}\mathbf{z} - c\mathbf{t}_\tau + c\mathbf{t}_{0,\tau}, 2\gamma)$
13 : return $(c, (\mathbf{z}, \mathbf{h}))$
```

Figure 2.4: Real and simulated transcripts of the blDilithium-QROM-ID protocol.

Lemma 4. *If* $\max_{s \in S_\eta, c \in \mathsf{ChSet}} \|2cs\|_\infty \leq \beta$, $\max_{t_0 \in S'_{2^d}, c \in \mathsf{ChSet}} \|2ct_0\|_\infty \leq \gamma$, $\beta \ll \gamma'$, *and* $\beta + 1 < 2\beta$, *then* blDilithium-QROM-ID *has correctness error*

$$\delta \approx 1 - \exp\left(-\beta n \left(\frac{\ell}{\gamma'} + \frac{k}{\gamma}\right)\right).$$

Proof. We begin by computing the probability Prv_1 or Prv_2 does not output $(\perp, (\perp, \perp))$. The probability $(\perp, (\perp, \perp))$ is not output in line 12 of Trans is simply the probability it is not output in line 7 of Sim, thus this probability is

$$\frac{\left|S^\ell_{\gamma'-\beta-1}\right|}{\left|S^\ell_{\gamma'-1}\right|} = \left(\frac{2(\gamma' - \beta) - 1}{2\gamma' - 1}\right)^{n\ell} > \left(1 - \frac{\beta}{\gamma'}\right)^{n\ell} \approx \exp\left(-\frac{\beta n \ell}{\gamma'}\right)$$

as $\beta \ll \gamma'$. On the assumption that the distribution of $\mathbf{A}\mathbf{z} - c(\mathbf{t} + \mathbf{t}') \mod 2\gamma$ is close to uniform when $\mathbf{z} \in S^k_{\gamma'-\beta-1}$ is uniformly sampled, then the probability $(\perp, (\perp, \perp))$ is not output in line 14 of Trans or equivalently line 11 of Sim is

$$\Pr[\mathbf{z} \leftarrow_{\$} S^{\ell}_{\gamma'-\beta-1} \mid \left\|\mathsf{LowBits}_q(\mathbf{Az} - c(\mathbf{t}+\mathbf{t}'))\right\|_\infty < \gamma - \beta] \approx \frac{\left|S^{k}_{\gamma-\beta-1}\right|}{\left|S^{k}_{\gamma-1}\right|} \approx \exp\left(-\frac{\beta n k}{\gamma}\right).$$

Hence

$$\Pr[\mathbf{y} \leftarrow_{\$} S^{\ell}_{\gamma'-1}, c \leftarrow_{\$} \mathsf{ChSet} \mid (\mathbf{z},\mathbf{h}) \neq (\perp,\perp)] \approx \exp\left(-\beta n\left(\frac{\ell}{\gamma'} + \frac{k}{\gamma}\right)\right).$$

Finally, assume $(\mathbf{z},\mathbf{h}) \neq (\perp,\perp)$. Then $\mathsf{blDilithium\text{-}QROM\text{-}ID.Verify}$ will always accept. Clearly, $\|\mathbf{z}\|_\infty < \gamma' - \beta$. Also, since

$$\mathbf{w} - c(\mathbf{s}_2 + \mathbf{s}_2') = \mathbf{Az} - c(\mathbf{t}_0 + \mathbf{t}_0') - c(\mathbf{t}_1 + \mathbf{t}_1') \cdot 2^d$$

and $\left\|c(\mathbf{t}_0 + \mathbf{t}_0')\right\|_\infty < \gamma$ and $\left\|\mathsf{LowBits}_q(\mathbf{w} - c(\mathbf{s}_2 + \mathbf{s}_2'), 2\gamma)\right\|_\infty < \gamma - \beta$, by Lemmas 4.1 and 4.2 of [103],

$$\mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c(\mathbf{t}_1 + \mathbf{t}_1') \cdot 2^d, 2\gamma) = \mathsf{HighBits}_q(\mathbf{w} - c(\mathbf{s}_2 + \mathbf{s}_2'), 2\gamma) = \mathbf{w}_1$$

Hence, $\mathsf{blDilithium\text{-}QROM\text{-}ID}$ has correctness error based solely off of the probability Prv fails, thus

$$\delta \approx 1 - \exp\left(-\beta n\left(\frac{\ell}{\gamma'} + \frac{k}{\gamma}\right)\right).$$

□

Lossiness

We now show that a bounded adversary has trouble distinguishing valid identity public keys as done in Fig 2.3 and randomly generated identity public keys as done in Fig 2.5. In addition, given a randomly generated identity public key, any unbounded adversary has only a little more than $1/|\mathsf{ChSet}|$ probability of impersonating the prover. More concretely, we address these two properties as defined in Definition 2.8 of [103].

Lemma 5. *For any adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{\mathsf{LOSS}}_{\mathsf{blDilithium\text{-}QROM\text{-}ID}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{k,\ell,\mathcal{U}}(\mathcal{A})$$

where $\mathcal{U}$ is the uniform distribution over S_η.

```
blDilithium-QROM-ID.LosKeyGen()
1 : $\mathbf{t} \leftarrow_\$ R_q^k$
2 : $\mathbf{t}_1 \leftarrow \mathsf{Power2Round}_q(\mathbf{t}, d)$
3 : $\mathbf{t}_0 \leftarrow \mathbf{t} - \mathbf{t}_1 \cdot 2^d$
4 : return $pk = (\mathbf{t}_1, \mathbf{t}_0)$
```

Figure 2.5: Lossy key generator of blDilithium-QROM-ID.

```
Game LOSSY-IMP
1 : $pk_{\mathsf{ls}} = (\mathbf{t}_1, \mathbf{t}_0) \leftarrow \mathsf{blDilithium\text{-}QROM\text{-}ID.LosKeyGen}()$
2 : $(\mathbf{w}_1, \tau, St) \leftarrow \mathcal{A}(pk_{\mathsf{ls}})$
3 : $c \leftarrow \mathsf{ChSet}$
4 : $(\mathbf{z}, \mathbf{h}) \leftarrow \mathcal{A}(St, c, \tau)$
5 : return $\mathsf{Verify}(pk_{\mathsf{ls}}, \mathbf{w}_1, c, (\mathbf{z}, \mathbf{h}))$
```

Figure 2.6: LOSSY-IMP game.

Proof. Differentiating between the output of blDilithium-QROM-ID.KeyGen and blDilithium-QROM-ID.LosKeyGen is clearly equivalent to differentiating between MLWE samples with static $\mathbf{A}$ and uniform samples over R_q^k. □

Lemma 6. *If* $q \equiv 5 \pmod 8$, $\max_{s \in S_\eta, c \in \mathsf{ChSet}} \|2cs\|_\infty \leq \beta$, $4\gamma + 2^d\beta + 2, 2\gamma' + (2^d - 2)\beta - 2 < \sqrt{q/2}$, *and* $2^d < 4\gamma' + (2^{d+1} - 4)\beta - 4$, *then* blDilithium-QROM-ID *has* $\varepsilon_{\mathsf{ls}}$*-lossy soundness for*

$$\varepsilon_{\mathsf{ls}} \leq \frac{1}{|\mathsf{ChSet}|} + 2|\mathsf{ChSet}|^2 \cdot \left(\frac{(4\gamma' + (2^{d+1} - 4)\beta - 3)^\ell \cdot (8\gamma + 2^{d+1}\beta + 5)^k}{q^k} \right)^n .$$

Proof. Let $\mathcal{A}$ be an unbounded adversary executed in the LOSSY-IMP game as shown in Fig 2.6. We first consider the case where there exist two distinct $(c, (\mathbf{z}, \mathbf{h})), (c', (\mathbf{z}', \mathbf{h}'))$ such that $\mathcal{A}$ is able to impersonate the prover. It follows that $\|\mathbf{z}\|_\infty, \|\mathbf{z}'\|_\infty < \gamma' - \beta$ and

$$\mathbf{w}_1 = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - (\mathbf{t}_1 + \mathbf{t}_1')c \cdot 2^d, 2\gamma) = \mathsf{UseHint}_q(\mathbf{h}', \mathbf{Az}' - (\mathbf{t}_1 + \mathbf{t}_1')c' \cdot 2^d, 2\gamma)$$

Thus by Lemma 4.1 of [103]

$$\left\| \mathbf{Az} - (\mathbf{t}_1 + \mathbf{t}_1')c \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma \right\|_\infty \leq 2\gamma + 1,$$

$$\left\| \mathbf{A}\mathbf{z}' - (\mathbf{t}_1 + \mathbf{t}_1')c' \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma \right\|_\infty \leq 2\gamma + 1.$$

So by the triangle equality,

$$\left\| \mathbf{A}(\mathbf{z} - \mathbf{z}') - (\mathbf{t}_1 + \mathbf{t}_1') \cdot (c - c') \cdot 2^d \right\|_\infty \leq 4\gamma + 2$$

Hence, for some $\mathbf{u}$ such that $\|\mathbf{u}\|_\infty \leq 4\gamma + 2$,

$$\mathbf{A}(\mathbf{z} - \mathbf{z}' - \mathbf{s}_1' \cdot 2^d(c - c')) + (\mathbf{u} - \mathbf{s}_2' \cdot 2^d(c - c')) = \mathbf{t}_1 \cdot 2^d(c - c').$$

Since $\|2c\mathbf{s}_1\|_\infty \leq \beta$, then $\left\|\mathbf{z} - \mathbf{z}' - \mathbf{s}_1' \cdot 2^d(c - c')\right\|_\infty \leq 2(\gamma' - \beta - 1) + 2^d\beta$ and $\left\|\mathbf{u} - \mathbf{s}_2' \cdot 2^d(c - c')\right\|_\infty \leq 4\gamma + 2 + 2^d\beta$, then by Lemma 4.6 of [103], the above equation is satisfied with probability

$$2|\mathsf{ChSet}|^2 \cdot \left(\frac{(4\gamma' + (2^{d+1} - 4)\beta - 3)^\ell \cdot (8\gamma + 2^{d+1}\beta + 5)^k}{q^k} \right)^n.$$

In the case where there is only one c that allows $\mathcal{A}$ to impersonate the prover, then $\mathcal{A}$ only has a $1/|\mathsf{ChSet}|$ probability of winning the LOSSY-IMP game. The Lemma follows from combining probabilities of both these cases. □

Min Entropy

We finally consider the probability that the $\mathbf{w}_1$ output is distinct for every run of the prover. As Prv_1 is identical between Dilithium-QROM-ID and blDilithium-QROM-ID, Lemma 4.7 of [103] applies directly to this setting. As such, we restate Lemma 4.7 here.

Lemma 7 (Lemma 4.7 of [103]). *If $2\gamma, 2\gamma' < \sqrt{q/2}$ and $\ell \leq k$ then* blDilithium-QROM-ID *has*

$$\alpha > n\ell \cdot \log \left(\min \left(\frac{q}{(4\gamma + 1)(4\gamma' + 1)}, 2\gamma' - 1 \right) \right)$$

bits of min-entropy as defined in Definition 2.6 of [103].

Unforgeability under Chosen Message and Epoch Attack

We now arrive at showing the unforgeability of blDilithium-QROM in the context of blinded signature schemes. First observe that $\mathsf{FS}[\text{blDilithium-QROM-ID}, \mathsf{H}, \frac{200}{1-\delta}]$ is equivalent to blDilithium-QROM′.

The following Theorem is a direct result of preceding Lemmas as well as the techniques used to prove Theorem 3.1 of [103].

Theorem 1. *Let $\mathcal{A}$ be any adversary that makes at most q_{H} hash queries and q_S signing queries against the unforgeability of* $\mathsf{blDilithium\text{-}QROM}'$ *with parameters as specified in Figure 2.2, with key generation min-entropy $2^{-h_{pk}}$. Then there exists an algorithm $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathsf{EUF-CMEA}}_{\mathsf{blDilithium\text{-}QROM}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{k,\ell,\mathcal{U}}(\mathcal{B}) + q_H q_S \cdot 2^{-h_{pk}} + 8(q_{\mathsf{H}}+1)\cdot 2^{-137} + q_S 2^{-2899}$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A}) + \kappa_m q_H q_S$.

Proof. We want to adapt Theorem 3.1 of [103] to our context of existential unforgeability under chosen message and epoch attack. To begin, we summarize the structure of the proof of Theorem 3.1. We note that this theorem is interested in establishing the quantum random oracle model security of the Fiat-Shamir transformation for lossy ID schemes. However because we are only interested in the classical random oracle model security, we use the classical version of the theorem. As the authors note, the only difference between the two bounds is whether one of the terms is linear or quadratic in q_H.

The proof of existential unforgeability is established in two steps: first they show that the lossiness property implies existential unforgeability with respect to a key-only adversary. They then show that if a scheme is secure against a key-only adversary, and also is naHVZK and the min-entropy of the commitment (as canonical ID scheme) has sufficient min-entropy, then the scheme is existentially unforgeable.

We now consider how these two steps must change to properly account for the additional powers we give the adversary leveraged by blinding. In the first step, we want to show that it is impossible for the adversary to create signatures for a lossy public key. The only modification that blinding introduces here is that the adversary is permitted to blind the identity public key however they would like, and then attempt to create a forgery with respect to that blinding. So, we work with a slightly altered version of lossiness to consider this. This is what we address with Lemma 6, showing that even with this additional degree of freedom, constructing forgeries for a lossy key is still impossible.

Thus making the same arguments as in [103], we have that

$$\mathbf{Adv}^{\mathsf{EU\text{-}CMEA}}_{\mathsf{blDilithium\text{-}QROM}}(\mathcal{A}^{\mathsf{KO}}) \leq \mathbf{Adv}^{\mathsf{LOSS}}_{\mathsf{blDilithium\text{-}QROM\text{-}ID}}(\mathcal{B}) + 8(q_H+1)\cdot \varepsilon_{\mathsf{ls}}. \tag{2.1}$$

As mentioned previously, this bound depends linearly on q_H, as we are working in the classical random oracle model.

By Lemma 5,

$$\mathbf{Adv}^{\mathsf{LOSS}}_{\mathsf{blDilithium\text{-}QROM\text{-}ID}}(\mathcal{B}) = \mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{k,\ell,\mathcal{U}}(\mathcal{B})$$

where $\mathcal{U}$ is the uniform distribution over S_η. By Lemma 6,

$$\varepsilon_{\mathsf{ls}} \leq \frac{1}{|\mathsf{ChSet}|} + 2|\mathsf{ChSet}|^2 \cdot \left(\frac{(4\gamma' + (2^{d+1} - 4)\beta - 3)^\ell \cdot (8\gamma + 2^{d+1}\beta + 5)^k}{q^k} \right)^n.$$

Substituting in the parameters from Figure 2.2 and noting that the magnitude of $|\mathsf{ChSet}|$ is $\binom{512}{46} \cdot 2^{46} > 2^{265}$, we get that $\varepsilon_{\mathsf{ls}} \leq 2^{-265} + 2^{-138} \leq 2^{-137}$.

The next step is to relate the EU-CMEA security to the EU-CMEA security against a key-only adversary. In [103] this is done in Theorem 3.3 by a game hopping argument and using the naHVZK property to simulate signatures. In Figures 2.7 and 2.8 we illustrate their proof, adapted to incorporate blinding. Note that line comments indicate which line of code is run in each game. If a line is unannotated, it is run in all four game hops. Game 0 is the plain EU-CMEA game, and so $\mathbf{Adv}^{\mathsf{EU\text{-}CMEA}}_{\mathsf{blDilithium\text{-}QROM}}(\mathcal{A})$.

In Game 1, we modify the how the random oracle and the blinding oracle interact with each other. Whenever the adversary queries $G(pk\|\tau)$, we answer honestly, returning (s_1', s_2'). If pk is the identity public key, we also calculate the corresponding pk_τ and add $(pk, \tau, pk_\tau = (\mathbf{t}_{1,\tau}, \mathbf{t}_{0,\tau})$ to a maintained table T. We then abort if the following sequence of events (denoted event E_1) happens:

1. The adversary queries $H(msg\|\mathbf{w}_1\|\mathbf{t}_{1,\tau})$, with $\mathbf{t}_{1,\tau}$ not appearing as part of any public key in the table T.
2. The adversary later makes a query $(pk\|\tau)$ to G where pk is the identity public key and this causes $\mathbf{t}_{1,\tau}$ to be added to T.

Since we are not changing any distributions or how oracles are run, we only need to consider the probability of event E_1 to bound the difference between these two games. E_1 essentially means that the adversary was able to 'guess' a blinded public key prior to actually querying to get that public key, so the probability of E_1 can be determined from the entropy of the blinding process. The entropy of the blinding process is straightforward to consider for Dilithium-QROM, as we simply sample a new secret key, generate the associated public key, and add it to the identity key. So the entropy of the blinded public key given the identity public key is simply the entropy of the plain key generation process. As a result we can see that $\Pr[E_1] \leq q_H q_S \cdot 2^{-h_{pk}}$.

In Game 2, the naHVZK property is used to simulate the signing oracle instead of properly signing messages. With blinding being introduced, we now require that the naHVZK property can hold with respect to any blinding that the adversary that the adversary might use for

```
Game Structure
1 : $(pk, sk) \leftarrow \mathsf{blDilithium\text{-}QROM'}.\mathsf{KeyGen}()$
2 : $(msg^*, \tau^*, \sigma^*) \leftarrow \mathcal{A}^{H,G,\mathsf{Sign}}(pk)$
3 : Parse $\sigma^* = (c^*, \mathbf{z}^*, \mathbf{h}^*)$
4 : $pk_{\tau^*} = (\mathbf{t}_{1,\tau^*}, \mathbf{t}_{0,\tau^*}) \leftarrow \mathsf{blDilithium\text{-}QROM'}.\mathsf{BlindPk}(pk, \tau^*)$
5 : $\mathbf{w}_1^* \leftarrow \mathsf{UseHint}_q(\mathbf{h}^*, \mathbf{A}\mathbf{z}^* - c^*\mathbf{t}_{1,\tau^*} \cdot 2^d, 2\gamma)$
6 : if $c^* \neq H'(msg \| \mathbf{w}_1^* \| pk_{\tau^*})$ then return 0    // $G_3$
7 : return $(msg^*, \tau^*) \notin \mathcal{M} \wedge \mathsf{blDilithium\text{-}QROM'}.\mathsf{Verify}(msg^*, \sigma^*, pk_{\tau^*})$

GetTrans$(msg, \tau, i)$
1 : $pk_\tau \leftarrow \mathsf{blDilithium\text{-}QROM'}.\mathsf{BlindPk}(pk, \tau)$
2 : $\kappa = 0$
3 : while $\mathbf{z} = \perp$ and $\kappa \leq \kappa_m$ do
4 :     $\kappa \leftarrow \kappa + 1$
5 :     $(\mathbf{w}_1, c, (\mathbf{z}, \mathbf{h})) \leftarrow \mathsf{Sim}(pk_\tau; \mathsf{RF}(msg \| pk_\tau \| \kappa \| i))$
6 : return $(\mathbf{w}_1, c, (\mathbf{z}, \mathbf{h}))$

Sign$(msg, \tau)$
1 : return $\mathsf{blDilithium\text{-}QROM}.\mathsf{Sign}(msg, pk, sk, \tau)$    // $G0, G1$
2 : $ctr_{msg} \leftarrow ctr_{msg} + 1$    // $G2, G3$
3 : $\mathcal{M} \leftarrow \mathcal{M} \cup \{(msg, \tau)\}$    // $G2, G3$
4 : $(\mathbf{w}_1, c, (\mathbf{z}, \mathbf{h}) \leftarrow \mathsf{GetTrans}(msg, \tau, \mathsf{ctr}_{msg})$    // $G2, G3$
5 : return $(c, (\mathbf{z}, \mathbf{h}))$    // $G2, G3$
```

Figure 2.7: Illustrating the game-hopping proof for reducing existential unforgeability to key-only security for blDilithium-QROM′.

their blinding query, which is what we prove in Lemma 3. Therefore, we rely on the same logic used to prove Theorem 3.3 of [103] to note that the difference between games 1 and 2 are at most $\kappa_m q_S \varepsilon_{\mathsf{zk}}$, where $\kappa_m = \frac{200}{1-\delta}$ is informed from the correctness error.

In more detail, we use the Sim procedure in Figure 2.4 to sign messages. For a (msg, τ) pair we define a GetTrans(msg, τ, ctr) oracle used to generate valid transcripts. This oracle operates by running the Sim procedure, with the randomness seeded by the string

```
$G(pk\|\tau)$
1: if $pk$ is the identity public key provided to $\mathcal{A}$   // $G1, G2, G3$
2:     // Use G' here to prevent recursion
3:     $pk_\tau \leftarrow \mathsf{blDilithium\text{-}QROM.BlindPk}(pk, \tau)$   // $G1, G2, G2$
4:     if $pk_\tau \in B$ abort   // $G1, G2, G2$
5:     $T \leftarrow T \cup \{(\tau, pk_\tau)\}$   // $G1, G2, G3$
6: return $G'(pk\|\tau)$

$H(\mathbf{w}_1\|msg\|pk_\tau)$
1: if $(\tau, pk_\tau) \notin T$   // $G1, G2, G3$
2:     $B \leftarrow B \cup \{pk_\tau\}$   // $G1, G2, G3$
3:     return $H'(\mathbf{w}_1\|msg\|pk_\tau)$   // $G2, G3$
4: for $i$ in $\{1, \dots, q_S\}$   // $G2, G3$
5:     $(\mathbf{w}_1', c, (\mathbf{z}, \mathbf{h})) \leftarrow \mathsf{GetTrans}(msg, \tau, i)$   // $G2, G3$
6:     if $\mathbf{w}_1' = \mathbf{w}_1$ return $c$   // $G2, G3$
7: return $H'(\mathbf{w}_1\|msg\|pk_\tau)$
```

Figure 2.8: Illustrating how random oracles are managed in game-hopping proof of Figure 2.7.

$msg\|pk_\tau\|\kappa\|ctr$, where ctr counts how many times the adversary has made this specific signing query and κ counts how many times Sim has failed to produce a transcript for this query. If the result is $(\bot, (\bot, \bot))$ then we increment κ and try again. Otherwise return $(\mathbf{w}_1, c, (\mathbf{z}, \mathbf{h}))$, where $\mathbf{w}_1 = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_{1,\tau} \cdot 2^d, 2\gamma)$. If after $\kappa_m = 200/(1-\delta)$ attempts we have failed to get a successful transcript, return $(\bot, \bot, (\bot, \bot))$.

We then can simulate the signing oracle as follows. When a query (msg, τ) is made for the ith time, generate $(\mathbf{w}_1, c, (\mathbf{z}, \mathbf{h})) \leftarrow \mathsf{GetTrans}(pk, \tau, i)$. Return $\sigma = (c, \mathbf{z}, \mathbf{h})$. The naHVZK property gives us that the distribution of the signature provided to the adversary has a statistical difference of at most $\kappa_m q_S \epsilon_{zk}$ from how signatures are meant to be truly generated.

However, we also need to make sure that the random oracle H is handled in such a way that these signatures actually verify. To do this, we use the GetTrans oracle to check and see if a query needs to be modified. We start with an 'unmodified' hash function H'. When the adversary makes a query of the form $msg\|\mathbf{w}_1\|pk_\tau$ to H, we use the lookup table

T to find the τ value that blinded pk to pk_τ. If pk_τ does not appear in T, then we simply rely on H' to answer the query. Otherwise, we check and see if the $\mathbf{w}_1$ part of the query matches with a possible signing query the adversary has made or might make, and if it is, return the c required to make it consistent.

Note that this will result in a consistent query response, unless the adversary makes a query $H(\mathbf{w}_1 \| msg \| pk_\tau)$ where (τ, pk_τ) is not in T, but then later it becomes added to T, for some τ, causing an inconsistency in the responses of H. But this is precisely the event E_1, and we have already considered its probability.

Finally, in Game 3, we return 0 if for the forgery submitted we have that $c^* \neq H'(msg \| w_1^* \| pk_{\tau^*})$. In other words, we return 0 if the forgery submitted is on a 'programmed' point of the random oracle. But this means there exists a value $1 \leq ctr \leq q_S$ with $\mathbf{w}_1^*$ matching $\mathbf{w}_1$ from $\mathsf{GetTrans}(msg^*, \tau^*, ctr)$, but $(msg^*, \tau^*) \notin \mathcal{M}$. But this means that this associated $\mathbf{w}_1$ has not been revealed as part of any signing query and no information about it has been revealed. So, for the adversary to "predict" the value, we need to consider its min-entropy, α. Following the logic in the original proof, we have that the difference in the success probabilities in Games 2 and 3 is at most $q_S 2^{-\alpha+1}$.

Finally, we need to consider the success probability in Game 3. Here, we note that no signing queries are being made. As well, if the adversary succeeds (and '1' is returned) in Game 3, we have that this is a valid forgery with respect to the hash function H'. In other words, we have made no signing queries (relying entirely on simulating transcripts and programming the random oracle), but have a forgery valid for an un-programmed oracle H'. Thus we can bound the success probability of the adversary in this game by the key-only success probability.

So, we have that

$$\mathbf{Adv}^{\mathsf{EU\text{-}CMEA}}_{\mathsf{blDilithium\text{-}QROM}}(\mathcal{A}) \leq q_H q_S \cdot 2^{-h_{pk}} + \kappa_m q_S \cdot \epsilon_{zk} + q_S \cdot 2^{1-\alpha} + \mathbf{Adv}^{\mathsf{EU\text{-}CMEA}}_{\mathsf{blDilithium\text{-}QROM}}(\mathcal{A}^{\mathsf{KO}}).$$

By Lemma 3, $\varepsilon_{\mathsf{zk}} = 0$. Finally, by Lemma 7, $2^{1-\alpha} < 2^{-2899}$. Combining this with Equation 2.1 we get the desired result.

□

2.5.4 blDilithium-QROM Unlinkability

We now proceed to showing that $\mathsf{blDilithium\text{-}QROM}'$ is unlinkable, and begin by demonstrating that it satisfies the independent blinding property.

Lemma 8. *If the SA-MWLE problem is hard, then* $\mathsf{blDilithium\text{-}QROM'}$ *satisfies independent blinding. In particular, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathsf{Ind-Blind}}_{\mathsf{blDilithium\text{-}QROM},t}(\mathcal{A}) \leq 2t\mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{m,k,\mathcal{U},\mathbf{A}}(\mathcal{B})$$

where $\mathcal{U}$ is the uniform distribution over S_η.

Proof. We begin by defining the $\mathsf{RandBlind}$ function for $\mathsf{blDilithium\text{-}QROM'}$ as identical to $\mathsf{BlindPk}$ except that $(\mathbf{s}'_1, \mathbf{s}'_2)$ are uniformly sampled from $S^\ell_\eta \times S^k_\eta$ rather than output from $\mathsf{G}(pk\|\tau)$.

Let G_0 be a game where $\mathcal{A}$ is given independent blindings of a single identity public key and game G_4 be a game where $\mathcal{A}$ is given independent blindings of independent identity public keys.

We first introduce a game G_1 which differs from G_0 in that each output of $\mathsf{RandBlind}$, specifically $\mathbf{t}_0 + \mathbf{t}'_i$ for $1 \leq i \leq t$ where $\mathbf{t}'_i$ is an SA-MLWE sample, is replaced with $\mathbf{t}_0 + \tilde{\mathbf{t}}'_i$ for $1 \leq i \leq t$ where $\tilde{\mathbf{t}}'_i$ is randomly and uniformly sampled from R^k_q. Note that any adversary $\mathcal{A}$ that can differentiate between a single sample $\mathbf{t}_0 + \mathbf{t}'$ and $\mathbf{t}_0 + \tilde{\mathbf{t}}'$ can be used to construct an adversary $\mathcal{B}$ that differentiates between plain SA-MLWE samples, as the SA-MLWE challenge can be transformed into an input $\mathcal{A}$ uses by simply adding $\mathbf{t}_0$ to the challenge. As we must replace each of the t samples of $\mathbf{t}_0 + \mathbf{t}'_i$ individually, the triangle equality implies that $\left|Pr[\mathcal{A}^{G_0} \Rightarrow 1] - Pr[\mathcal{A}^{G_1} \Rightarrow 1]\right| \leq t\mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{m,k,\mathcal{U},\mathbf{A}}(\mathcal{B})$.

We now introduce G_2, which is identical to G_1 except that each sample $\mathbf{t}_0 + \tilde{\mathbf{t}}'_i$ for $1 \leq i \leq t$ is replaced with a sample $\tilde{\mathbf{t}}_i$ where $\tilde{\mathbf{t}}_i$ is randomly and uniformly sampled from R^k_q. Recall that $\tilde{\mathbf{t}}'_i$ is randomly and uniformly sampled from R^k_q, thus it is effectively randomly permuting $\mathbf{t}_0$ over R^k_q so G_1 and G_2 are indistinguishable.

In G_3, we now take the reverse step of replacing each $\tilde{\mathbf{t}}_i$ for $1 \leq i \leq t$ in G_2 with $\mathbf{t}_i + (\tilde{\mathbf{t}}_i - \mathbf{t}_i)$. Note $\tilde{\mathbf{t}}_i - \mathbf{t}_i$ is uniformly random over R^k_q. G_3 is clearly equivalent to G_2.

Finally, we make the hop from G_3 to G_4 by replacing each $\mathbf{t}_i + (\tilde{\mathbf{t}}_i - \mathbf{t}_i)$ with $\mathbf{t}_i + \mathbf{t}'_i$ where $\mathbf{t}'_i$ is an MLWE sample for $1 \leq i \leq t$. As $\tilde{\mathbf{t}}_i - \mathbf{t}_i$ is uniformly random over R^k_q, the bound in the difference between G_3 and G_4 is the same as the bound in the difference between G_0 and G_1. Hence by the triangle equality,

$$\left|Pr[\mathcal{A}^{G_0} \Rightarrow 1] - Pr[\mathcal{A}^{G_4} \Rightarrow 1]\right| = \mathbf{Adv}^{\mathsf{Ind-Blind}}_{\mathsf{blDilithium\text{-}QROM},t}(\mathcal{A}) \leq 2t\mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{m,k,\mathcal{U},\mathbf{A}}(\mathcal{B}).$$

□

We now show that $\mathsf{blDilithium\text{-}QROM}'$ permits signing with oracle reprogramming as specified in Definition 4. Let Forge output $(y = c, \sigma = (\mathbf{z}, \mathbf{h}, c))$ where $\mathbf{z}$, $\mathbf{h}$, and c are output from Sim in Figure 2.4. In addition, let Ext output $M \| \mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_{1,\tau} \cdot 2^d, 2\gamma) \| \mathbf{t}_{1,\tau}$. Since by Lemma 3 $\mathsf{blDilithium\text{-}QROM}'$ is naHVZK given appropriate parameters, then real signatures and the output of Forge are drawn from the exact same distributions, thus $\delta = 0$. Furthermore, the min entropy of Ext is solely dependent on $\mathbf{w}_1 = \mathbf{Az} - c\mathbf{t}_{1,\tau} \cdot 2^d$, as the adversary has control over M and knowledge of $\mathbf{t}_{1,\tau}$, hence the min entropy of Ext is the same as the min entropy from Lemma 7.

We are now ready to consider unlinkability under chosen message and epoch attack, which follows directly from the preceding Lemmas as well as the Lemmas in Section 2.4.

Theorem 2. *For any adversary $\mathcal{A}$ that makes q_S signing queries and q_{H} random oracle queries, there exists an algorithm $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathsf{UL-CMEA}}_{\mathsf{blDilithium\text{-}QROM},t}(\mathcal{A}) \leq 4t\mathbf{Adv}^{\mathsf{SA\text{-}MLWE}}_{m,k,\mathcal{U},\mathbf{A}}(\mathcal{B}_1) + q_{\mathsf{H}}\mathbf{Adv}^{\mathsf{EUF-CMEA}}_{\mathsf{blDilithium\text{-}QROM}}(\mathcal{B}_2) + q_{\mathsf{H}}q_S 2^{-2899}.$$

2.5.5 Key-blinding DILITHIUM

We briefly describe a key-blinded version of Dilithium [65] but we provide no security analysis or guarantees.

As is with blDilithium-QROM, $\mathbf{A}$ is a public parameter of the network and thus ρ can be omitted from the scheme. In addition, Power2Round is modified to release one extra bit for $\mathbf{t}_1$ while keeping $\mathbf{t}_0$ the same. The appropriate changes to Sign and Verify are made in a similar fashion to the changes made from Dilithium-QROM to blDilithium-QROM.

Note that during signing, tr may be recomputed as it is dependent solely on the identity public key and not the blinded public key. One possibility could be to set $tr = \mathsf{CRH}(\mathbf{s}_{1,\tau}, \|\mathbf{s}_{2,\tau})$.

No parameters need to be changed to modify the correctness of the blinded scheme.

2.6 An Isogeny-Based Key-Blinding Scheme

In this section we briefly describe how to realize a key-blinding signature scheme from CSI-FiSh [38], which is an isogeny-based signature scheme that uses the structure of the CSIDH [49] group action. The 'group' here refers to class group $\mathrm{Cl}(\mathcal{O})$, with $\mathcal{O}$ being the endomorphism ring $\mathrm{End}_{\mathbb{F}_p}(E)$, the ring of endomorphisms from a curve E to itself defined over $\mathbb{F}_p$, which is an order in the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$. A main contribution of the CSI-FiSh paper was to calculate the precise structure of this group, so that it can be described as a cyclic group of order N. This allows for two crucial operations with respect to the group action: group elements can now be sampled *uniformly* from the group, and group elements can now be given a *canonical representation* as a member of $\mathbb{Z}_N$, so that for example, when revealing to an adversary a group element $g = g_1 \cdot g_2$, we can be assured that no information about g_1 or g_2 is leaked by how g is represented.

For our purpose, we will describe the scheme as an abstract group action, and avoid notation that refers to how the group is actually constructed. For complete details about the group action we refer to the CSI-FiSh paper [38].

We briefly recall the details of a group action. We have a group G and a set $\mathcal{E}$ along with an operation $\star : G \times \mathcal{E} \to \mathcal{E}$. The operation $\star$ satisfies the property that if $id \in G$ is the identity group element, then $id \star E = E$ for all $E \in \mathcal{E}$. Furthermore, for $g_1, g_2 \in G$, it must be the case that $g_1 \star (g_2 \star E) = (g_1 \cdot g_2) \star E$. In fact, the group action described in [38] is both *free* and *transitive*, meaning that for $E_1, E_2 \in \mathcal{E}$ there is one and only one $g \in G$ such that $g \star E_1 = E_2$. Furthermore, the group G in our case is cyclic, and we will denote the order N.

In CSI-Fish, signatures correspond to a zero-knowledge proof of knowledge of a secret group element g_{sk} such that $g_{sk} \star E_0 = E_{pk}$, with E_{pk} being the public key and E_0 being a system parameter. Proving knowledge of such a g_{sk} is done via a simple sigma protocol. The commitment is created by uniformly sampling $g \leftarrow_{\$} G$ and computing $E_{com} = g \star E_{pk}$ as the commitment. The verifier then selects a bit $b \leftarrow_{\$} \{0, 1\}$ as the challenge, and the prover responds by sending g if $b = 0$, and $g \cdot g_{sk}$ if $b = 1$.

The verifier then checks: if $b = 0$ that $g \star E_{pk} = E_{com}$; and if $b = 1$ that $(g \cdot g_{sk}) \star E_0 = E_{com}$. Soundness follows from the fact that, from two responses g and $g \cdot g_{sk}$, the secret key g_{sk} can quickly be recovered. Honest-verifier zero-knowledge can be shown by simulating transcripts in a straightforward way (here we rely on the fact that group elements have a canonical representation).

The basic idea of how key-blinding functionality can be added to the scheme is already apparent. From a value τ, a group element g_τ can be generated, and the public key E_{pk} is blinded to $E_\tau = g_\tau \star E_{pk}$. Anyone who knows the public key and τ can perform this operation, but to sign a message, one must know g_τ and g_{sk} so the scheme is still unforgeable. Furthermore, because the group action is transitive, the action of g_τ entirely hides E_{pk}. Observing many blindings still leaks no information about E_{pk}, ensuring that the scheme is unlinkable.

Of course, the soundness of this zero-knowledge scheme is only 1/2, and would have to

```
blCSI-FiSh.KeyGen()
1 :  for $i \in [L]$ do
2 :      $g_{sk,i} \leftarrow\$ \mathbb{Z}_N$
3 :      $E_{pk,i} \leftarrow g_{sk,i} \star E_0$
4 :  endfor
5 :  return $(pk, sk) = ((E_{pk,i})_{i\in[L]}, (g_{sk,i})_{i\in[L]})$
```

```
blCSI-FiSh.BlindPk($(E_{pk,i})_{i\in[L]}, \tau$)
1 :  $(g_{\tau,i})_{i\in[L]} \leftarrow KDF((E_{pk,i})_{i\in[L]} \| \tau)$
2 :  for $i \in [L]$ do
3 :      $E_{\tau,i} \leftarrow g_{\tau,i} \star E_{pk,i}$
4 :  endfor
5 :  return $pk_\tau = (E_{\tau,i})_{i\in[L]}$
```

Figure 2.9: Key generation and blinding algorithms for blCSI-FiSh signature scheme.

be repeated many times in order for the signature scheme to be existentially unforgeable. The authors of CSI-FiSh employed many clever techniques in order to improve on the efficiency of the scheme over just repeating the signature scheme 128 times. Most notably, the public keys of CSI-FiSh consist of many curves $E_{pk,1}, E_{pk,2}, \ldots, E_{pk,L}$, generated by computing $g_{sk,1} \star E_0$, $g_{sk,2} \star E_0$, etc. Then rather than choosing a single bit for the challenge, an index from 0 to L can be chosen. This increases the soundness significantly, and so the protocol can be repeated fewer times to achieve the same level of security, allowing for a trade-off between the signature size and the public key size. To blind, we can similarly sample independent blinding factors $g_{\tau,1}, g_{\tau,2}, \ldots$ and apply each of them to each part of the public key.

CSI-FiSh uses a further technique to then double the number of available elliptic curves in the public key without increasing the corresponding size of the public key. For each curve $E_{pk,i}$, anyone can compute the quadratic twist of the curve, $E^t_{pk,i}$, and the associated secret key is $-g_{sk,i} \pmod N$. So if the public key contains L elliptic curves, the verifier can expand this to $2L$ curves, and we choose an index from $-L$ to L. To account for this in signing, we simply set $g_{sk,-i} = -g_{sk,i}$ for each $i \in [L]$.

The last optimization that CSI-FiSh optionally takes is to then 'Merkleize' the public key. Rather than including each of $E_{pk,1}, \ldots, E_{pk,L}$, key generation commits to these public keys by constructing a Merkle tree with each curve as a leaf node. When signing a message, each $E_{pk,i}$ that gets used, as well as the Merkle path that proves the commitment, is provided. This causes the public key to be only 32 bytes, at the expense of increasing the size of signatures and making signing and verification slightly slower. Unfortunately, it is not possible to use this technique for a blinded version. The raw $E_{pk,i}$ values must be available in order to construct the blinded version of the public key, and so 'Merkleization' is impossible.

Next, we describe signing and verification, which are essentially unchanged from in CSI-FiSh. We also prove the unlinkability of the scheme, and discuss the proof of unforgeability of the scheme. The greatest benefit of the CSI-FiSh scheme is that it requires the fewest changes to signing and verification. In this sense it is the most similar to the Ed25519 key-blinding scheme, where the public and secret keys are blinded in a straightforward way and signing and verification need not change. In Figure 2.10 we describe the signing and verification procedures. Note that except for the fact that the signing procedure starts by blinding the public key and incorporating the blinding factor into the secret key, it is not changed from in CSI-FiSh.

2.6.1 Unlinkability

To show that the scheme satisfies unlinkability, we need to prove the independent blinding property and the signing with oracle reprogramming property. We begin with the easier of the two, independent blinding. CSI-FiSh in fact satisfies the strongest possible independent blinding, as the adversary's advantage in distinguishing the two distributions is statistically 0. This is because the the distribution of $\mathsf{RandBlind}(pk, r)$ for a uniform r is actually uniform over the entire public key space and independent of pk.

This fact comes from the group action structure. Because we are using a regular group action, the act of sampling a group element $g_{\tau,i} \leftarrow_{\$} \mathbb{Z}_N$ and applying it to $E_{pk,i}$ provides a uniformly random element of the set, the distribution of which is thus independent of $E_{pk,i}$. So the act of blinding each entry in the public key with a uniform group element causes the entire public key to be perfectly uniformly random and independent, and thus the adversary's advantage in distinguishing the two distributions is 0.

For the signing with oracle reprogramming property, the proof is again a relatively straightforward consequence of the Fiat–Shamir paradigm. By choosing the indices $(c_1, \ldots, c_M)$ that will be returned from H in advance, we can easily construct a signature that will verify by selecting $r_i \leftarrow_{\$} \mathbb{Z}_N$ and computing $E_i \leftarrow r_i \star E_{c_i}$. It is easy to check the correctness and to see that the fact that our group action is free and transitive will guarantee that the distribution of the signatures generated is statistically identical. Furthermore, the $\mathsf{Extract}$ function will return the point $E_1 || \ldots || E_M || msg$, and as the E_i have a distribution uniform over the set (which has size roughly 2^{256}), the resulting min-entropy is 2^{256M}, more than sufficient.

```
blCSI-FiSh.Sign($msg, pk = (E_{pk,i})_{i\in[L]}, sk = (g_{sk,i})_{i\in[L]}, \tau$)
1 :  $(g_{\tau,i})_{i\in[L]} \leftarrow KDF((E_{pk,i})_{i\in[L]}||\tau)$
2 :  for $i \in [L]$ do
3 :      $g_{\tau,i} \leftarrow g_{\tau,i} + g_{sk,i} \pmod N$
4 :      $g_{\tau,-i} \leftarrow -g_{\tau,i} \pmod N$
5 :  endfor
6 :  $g_{\tau,0} \leftarrow 0$
7 :  for $i \in [M]$ do
8 :      $b_i \leftarrow_{\$} \mathbb{Z}_N$
9 :      $E_i \leftarrow b_i \star E_0$
10 : endfor
11 : $(c_1, \ldots, c_M) \leftarrow H(E_1||E_2|| \ldots E_M||msg), c_i \in \{-L, \ldots, L\}.$
12 : for $i \in [M]$ do
13 :     $r_i \leftarrow b_i - g_{\tau,c_i} \pmod N$
14 : endfor
15 : return $\sigma = (r_1, \ldots, r_M, c_1, \ldots, c_M)$

blCSI-FiSh.Verify($msg, blpk = (E_{\tau,i})_{i\in[L]}, \sigma = (r_1, \ldots, r_M, c_1, \ldots, c_M)$)
1 :  Let $E_{\tau,-i} = E^t_{\tau,i}$, the quadratic twist
2 :  for $i \in [M]$ do
3 :      $E_i \leftarrow r_i \star E_{\tau,c_i}$
4 :  endfor
5 :  if $(c_1, \ldots, c_M) = H(E_1|| \ldots ||E_M||msg)$ then return accept
6 :  else return reject
```

Figure 2.10: Signing and Verification algorithms for blCSI-FiSh signature scheme.

2.6.2 Unforgeability

The proof of unforgeability is straightforward and not particularly impacted by the addition of blinding. In the CSI-FiSh paper [38], the security of the scheme is proven by showing that the scheme satisfies special soundness, unique responses, large min entropy and satisfies honest-verifier zero knowledge. None of these proofs are changed by the addition of the blinding factor. The blinding factor merely introduces an arbitrary 'offset' for each of the parts of the public key, but this offset does not materially change anything about the security of the scheme, even if an adversary knows the offset.

2.7 Extending LegRoast with Key-Blinding

LegRoast and PorcRoast are new adaptations of Picnic that use the Legendre symbol as a symmetric PRF [34, 58]. In this section we show how the mathematical structure of LegRoast enables a more efficient key-blinding signature scheme compared to naïvely using the MPC-in-the-head framework.

Recall that the Legendre symbol modulo a prime p, denoted $(\frac{a}{p})$, is defined as 0 if $a \equiv 0 \pmod{p}$, 1 if a is a quadratic residue modulo p, and -1 if it is not. To use the Legendre symbol as a single-bit keyed PRF with input X and key K, we can define a function that returns values in $\{0, 1\}$. For an odd prime p, define $\mathcal{L}_K(X)$ to return 0 if $K + X$ is a quadratic residue or $0 \pmod{p}$, and 1 otherwise. This concept can be generalized to consider the ℓ-th power residue, instead of just quadratic residues. This allows for a keyed PRF with $\log_2 \ell$ bits of output to be defined as

$$\mathcal{L}_K^\ell(X) = \begin{cases} i, & \text{if } (X + K)/g^i \equiv h^\ell \pmod{p} \text{ for some } h \in \mathbb{F}_p^\times \\ 0, & \text{if } K + X \equiv 0 \pmod{p}. \end{cases}$$

A key property of this PRF is that it is a group homomorphism from $\mathbb{F}_p^\times$ to $\mathbb{Z}_\ell$. This is helpful for proving statements in zero-knowledge about preimages of the PRF. To prove knowledge of a K such that $\mathcal{L}_K^\ell(X) = s$, one can sample a random value $r \in \mathbb{F}_p^\times$ and send $(K + X) \cdot r$ and $\mathcal{L}_0^\ell(r)$. The prover then only needs to prove that the multiplication of $(K + X) \cdot r$ was computed correctly for the verifier to calculate s and be convinced of knowledge of K.[1] Since the equation being proven consists of a single multiplication gate, the resulting proof can be comparatively short.

[1]The verifier must also be convinced that the prover did not lie about the value of $\mathcal{L}_0^\ell(r)$. This is accomplished by having the prover commit to this value before the challenge X is issued, so that the prover cannot choose the output of the PRF in a way to help them.

LegRoast and PorcRoast [34] expand this idea into a signature scheme that uses the Fiat–Shamir heuristic. In LegRoast, the 1-bit Legendre PRF is used, while PorcRoast uses the generalized $\log_2 \ell$-bit output. In general, we do not distinguish between the two, and present the scheme with the general $\log_2 \ell$-bit PRF. Public keys consist of the output of L computations of the Legendre PRF, with inputs $\mathcal{I} = i_1, \ldots, i_L$, which can be public parameters. We define the function F, which is parameterized by ℓ and $\mathcal{I}$, as taking in the secret key K and returning $\mathcal{L}_K^\ell(i_m)$ for $m \in [L]$. Hence, key generation consists of sampling a random secret key $K \in \mathbb{F}_p^\times$ and computing the public key $F_\mathcal{I}^\ell(K) = \big(\mathcal{L}_K^\ell(i_1), \ldots, \mathcal{L}_K^\ell(i_L)\big)$.

The same homomorphic property that makes the Legendre symbol an attractive option for zero knowledge proofs is also what allows for a blinding mechanism. Hashing the nonce and public key to a value $T \in \mathbb{F}_p^\times$, we can calculate L computations of the Legendre PRF with separate inputs $j_1, \ldots, j_L$. The public key blinded under the value T becomes

$$\left(\mathcal{L}_K^\ell(i_1) + \mathcal{L}_T^\ell(j_1), \ldots, \mathcal{L}_K^\ell(i_L) + \mathcal{L}_T^\ell(j_L)\right),$$

where addition is performed modulo ℓ. Due to the homomorphic property of $\mathcal{L}$, this can also be written as $\big(\mathcal{L}_0^\ell((K + i_m) \cdot (T + j_m))\big)_{m \in [L]}$.

As mentioned, LegRoast works by presenting parts of the public key multiplied by random values $r^{(j)} \in \mathbb{F}_p^\times$, the results of which are denoted by $o^{(j)}$. Then the signer proves knowledge of K by presenting a zero knowledge proof that a random linear combination of B $(K + I^{(j)}) \cdot r^{(j)} - o^{(j)}$ terms is equal to 0; here the $I^{(j)}$ values are a random re-indexing of the $i^{(j)}$ values in the public key. We call such a linear combination the error term, which should be equal to zero. Once the coefficients $\{\lambda^{(j)}\}$ of the linear combination are defined, the error term is

$$E = \sum_{j=1}^{B} \lambda^{(j)} \left((K + I^{(j)}) \cdot r^{(j)} - o^j\right) = K \cdot \left(\sum_{j=1}^{B} \lambda^{(j)} r^{(j)}\right) + \sum_{j=1}^{B} \lambda^{(j)} (I^{(j)} r^{(j)} - o^{(j)}).$$

Since only the K and $r^{(j)}$ values are secret, the only time we have a secret value multiplied by a secret value is in the $K \cdot \sum \lambda^{(j)} r^{(j)}$ term, so the error term can be verified to be 0 with only one multiplication gate.

If we are using a blinded public key, then the corresponding error term is the summation of

$$\lambda^{(j)}((K + I^{(j)})(T + J^{(j)}) r^{(j)} - o^{(j)})$$

terms. Through rearranging in a similar way to that of LegRoast, we get an error term that has three multiplication gates as opposed to one. First, we discuss the blinding process.

```
blLegRoast.KeyGen()
1 : $K \leftarrow\!\!\$\, \mathbb{F}_p$
2 : for $i \in [L]$ do
3 :     $w_{i,pk} \leftarrow \mathcal{L}_T^{\ell}(i_i)$
4 : endfor
5 : return $(pk, sk) = (w_{i,pk})_{i\in[L]}, K)$

blLegRoast.BlindPk($(w_{i,pk})_{i\in[L]}, \tau$)
1 : $T \leftarrow KDF(pk||\tau)$
2 : for $i \in [L]$ do
3 :     $v_i \leftarrow \mathcal{L}_T^{\ell}(j_i)$
4 :     $w_{i,\tau} \leftarrow w_{i,pk} + v_i \pmod{\ell}$
5 : endfor
6 : return $pk_\tau = (w_{i,\tau})_{i\in[L]}$
```

Figure 2.11: blLegRoast.KeyGen and blLegRoast.BlindPk.

To complete the security assessment for blLegRoast, we still need to establish (i) the independent blinding property, (ii) the signing with oracle reprogramming property, and (iii) the existential unforgeability of the scheme. As the scheme uses the Fiat–Shamir heuristic, signing with reprogramming is possible by choosing the output of the hash function in advance and constructing the signature accordingly. The existential unforgeability of the scheme follows from how finding a K and T that satisfy the relations informed by the public key is still hard. In the remainder of this section, we provide a complete description of the scheme and the proofs of the unlinkability and unforgeability properties.

2.7.1 Algorithm Description

Here we detail the blinded version of LegRoast or PorcRoast. The scheme mostly resembles the original, but with a few key changes. For the blinded version, we are proving knowledge of a pair $(K, T) \in \mathbb{F}_p$ such that $F_{\mathcal{I}}^{\ell}(K) + F_{\mathcal{J}}^{\ell}(T) = pk_\tau$ (more accurately, we are proving a relaxation of this relation, but more on this later). This is done through proving that the error term from equation 2.2 is equal to zero. LegRoast's error term is very simple, consisting of a single multiplication gate. Our blinded version requires three. This increases the signature length, but also requires some revisions to how the signature scheme works. Other than simple modular arithmetic, LegRoast relies on a small number of symmetric primitives: hash functions $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$, a commitment hash function H_{com}, a hash function used to build a binary tree from a root value, and a Expand function that can be used to generate an arbitrary number of values in $\mathbb{F}_p$.

$$\begin{aligned}E &= \sum_{j=1}^{B} \lambda^{(j)} \left((K + I^{(j)}) \cdot (T + J^{(j)}) \cdot r^{(j)} - o^{(j)} \right) \\ &= K \cdot \left(T \cdot \left(\sum_{j=1}^{B} \lambda^{(j)} r^{(j)} \right) + \sum_{j=1}^{B} \lambda^{(j)} r^{(j)} J^{(j)} \right) \\ &\quad + T \cdot \left(\sum_{j=1}^{B} \lambda^{(j)} I^{(j)} r^{(j)} \right) + \sum_{j=1}^{B} \lambda^{(j)} \left(I^{(j)} J^{(j)} r^{(j)} - o^{(j)} \right) \end{aligned} \tag{2.2}$$

One issue is that because LegRoast has a single multiplication gate, the output of the gate is publicly known. This allows for some optimizations. As our blinded version has multiple nested multiplication gates, we need to more closely follow the zero-knowledge protocol from [20] that LegRoast uses. This means that the shares of the output gates z are not known, and a correction term Δz must be committed to by the prover. But since the indices $I_e^{(j)}$ are not known until after the prover has committed to the seeds, the correction values Δz are known in a round after the other correction values. To fix this issue, we insert an additional round for the prover. After the λ challenge has been generated, the prover must commit to the Δz correction terms before they get the ϵ challenge values. Other than this, our protocol largely follows that of LegRoast, except with the added values for the two additional multiplication gates used in the computation.

```
$\mathsf{blLegRoast.Sign}(msg, pk, sk, \tau)$, Part 1

    // Prover Part 1: Generating Pre-processing triples and input shares
1 : $T \leftarrow KDF(pk||\tau)$
2 : Pick a random $salt \leftarrow\!\$\ \{0,1\}^{2\lambda}$.
3 : for $e \in [M]$ do
4 :     Sample root seed $sd_e \leftarrow\!\$\ \{0,1\}^{\lambda}$.
5 :     Build a binary tree from $sd_e$ with leaves $sd_{e,1}, \ldots, sd_{e,N}$.
6 :     for $i \in [N]$ do
7 :         Sample shares: $\left\{\begin{array}{c} K_{e,i}, T_{e,i}, \\ r^{(1)}_{e,i}, \ldots, r^{(B)}_{e,i}, \\ a^{(1)}_{e,i}, a^{(2)}_{e,i}, a^{(3)}_{e,i}, \\ b^{(1)}_{e,i}, b^{(2)}_{e,i}, b^{(3)}_{e,i}, \\ c^{(1)}_{e,i}, c^{(2)}_{e,i}, c^{(3)}_{e,i}, \\ z^{(1)}_{e,i}, z^{(2)}_{e,i}, z^{(3)}_{e,i} \end{array}\right\} \leftarrow \mathsf{Expand}(sd_{e,i})$.
8 :         Commit to seed: $\mathsf{com}_{e,i} \leftarrow H_{com}(salt, e, i, sd_{e,i})$
9 :     endfor
10 :    Compute witness offsets: $\Delta K_e \leftarrow K - \sum_{i=1}^{N} K_{e,i},\ \ \Delta T_e \leftarrow T - \sum_{i=1}^{N} T_{i,e}$
11 :    Adjust first shares: $K_{e,1} \leftarrow K_{e,1} + \Delta K_e,\ \ T_{e,1} \leftarrow T_{e,1} + \Delta T_e$.
12 :    for $k \in [3]$ do
13 :        Compute triple: $a^{(k)}_e \leftarrow \sum_{i=1}^{N} a^{(k)}_{e,i},\ \ b^{(k)}_e \leftarrow \sum_{i=1}^{N} b^{(k)}_{e,i},\ \ c^{(k)}_e \leftarrow a^{(k)}_e \cdot b^{(k)}_e$
14 :        Compute triple offset: $\Delta c^{(k)}_e \leftarrow c^{(k)}_e - \sum_{i=1}^{N} c^{(k)}_{e,i}$
15 :        Adjust first share: $c^{(k)}_{e,1} \leftarrow c^{(k)}_{e,1} + \Delta c^{(k)}_e$.
16 :    endfor
17 :    for $j \in [B]$ do
18 :        $r^{(j)}_e \leftarrow \sum_{i=1}^{N} r^{(j)}_{e,i}$
19 :        Compute residue symbol: $s^{(j)}_e \leftarrow \mathcal{L}^{\ell}_0(r^{(j)}_e)$.
20 :    endfor
21 : endfor
22 : Set $\sigma_1 \leftarrow \left((\mathsf{com}_{e,i})_{i\in[N]}, (s^{(j)}_e)_{j\in[B]}, \Delta K_e, \Delta T_e, (\Delta c^{(k)}_e)_{k\in[3]}\right)_{e\in[M]}$.
```

Figure 2.12: $\mathsf{blLegRoast.Sign}$, Part 1 of 4.

```
$\mathsf{blLegRoast.Sign}(msg, pk, sk, \tau)$, Part 2

    // Challenger part 1: Issuing public key index challenge
23: Compute challenge hash: $h_1 \leftarrow \mathcal{H}_1(msg, salt, \sigma_1)$.
24: Expand hash: $(I_e^{(j)}, J_e^{(j)})_{e\in[M],j\in[B]} \leftarrow \mathsf{Expand}(h_1)$.
    // Prover part 2: Compute public product values
25: for $e \in [M]$ do
26:   for $j \in [B]$ do
27:     Compute output value: $o_e^{(j)} \leftarrow (K + I_e^{(j)})(T + J_e^{(j)}) \cdot r_e^{(j)}$.
28:   endfor
29: endfor
30: Set $\sigma_2 \leftarrow (o_e^{(j)})_{e\in[M],j\in[B]}$.
    // Challenger part 2: Linear combination challenge
31: Compute challenge hash: $h_2 \leftarrow \mathcal{H}_2(h_1, \sigma_2)$.
32: Expand the hash $(\lambda_e^{(1)}, \lambda_e^{(2)}, \ldots, \lambda_e^{(B)})_{e\in[M]} \leftarrow \mathsf{Expand}(h_2)$.
    // Prover part 3: Constructing output shares for multiplication gates
33: for $e \in [M]$ do
    // Compute first multiplication gate offset:
34:   $\Delta z_e^{(1)} \leftarrow T \cdot \left(\sum_j \lambda_e^{(j)} r_e^{(j)}\right) - \sum_i z_{e,i}^{(1)}$
    // Compute second multiplication gate offset:
35:   $\Delta z_e^{(2)} \leftarrow K \cdot \left(\left(\sum_j \lambda_e^{(j)} J_e^{(j)} r_e^{(j)}\right) + T \cdot \sum_j \lambda_e^{(j)} r_e^{(j)}\right) - \sum_i z_{e,i}^{(2)}$
    // Compute final multiplication gate offset:
36:   $\Delta z_e^{(3)} \leftarrow T \cdot \left(\sum_j \lambda_e^{(j)} I_e^{(j)} r_e^{(j)}\right) - \sum_i z_{e,i}^{(3)}$.
37:   for $k \in [3]$ do
38:     Adjust first shares: $z_{e,1}^{(k)} \leftarrow z_{e,1}^{(k)} + \Delta z_e^{(k)}$.
39:   endfor
40: endfor
41: Set $\sigma_3 \leftarrow (\Delta z_e^{(1)}, \Delta z_e^{(2)}, \Delta z_e^{(3)})_{e\in[M]}$.
```

Figure 2.13: blLegRoast.Sign, Part 2 of 4.

```
blLegRoast.Sign($msg, pk, sk, \tau$), Part 3

     // Challenger part 3: Multiplication gate challenge
42 : Compute challenge hash: $h_3 \leftarrow \mathcal{H}_3(h_2, \sigma_3)$.
43 : Expand the hash $(\epsilon_e^{(1)}, \epsilon_e^{(2)}, \epsilon_e^{(3)})_{e \in [M]} \leftarrow \mathsf{Expand}(h_3)$.
     // Prover part 4: Committing to the views of the MPC parties
44 : for $e \in [M]$ do
     // Views for first multiplication gate
45 :   for $i \in [N]$ do
46 :     Compute shares: $\alpha_{i,e}^{(1)} \leftarrow a_{e,i}^{(1)} + \epsilon_e^{(1)} \cdot T_{e,i}$,
47 :     $\beta_{e,i}^{(1)} \leftarrow b_{e,i}^{(1)} + \sum_j \lambda_e^{(j)} r_{e,i}^{(j)}$.
48 :   endfor
49 :   Compute values $\alpha_e^{(1)} \leftarrow \sum_i \alpha_{e,i}^{(1)}$ and $\beta_e^{(1)} \leftarrow \sum_i \beta_{e,i}^{(1)}$.
50 :   for $i \in [N]$ do
51 :     Compute $\gamma_{i,e}^{(1)} \leftarrow \epsilon_e^{(1)} z_{e,i}^{(1)} - c_{e,i}^{(1)} + \alpha_e^{(1)} b_{e,i}^{(1)} + \beta_e^{(1)} a_{e,i}^{(1)}$.
52 :     if $i = 1$ then set $\gamma_{e,i}^{(1)} \leftarrow \gamma_{e,i}^{(1)} - \alpha_e^{(1)} \cdot \beta_e^{(1)}$ endif .
53 :   endfor
     // Views for second multiplication gate
54 :   for $i \in [N]$ do
55 :     Compute shares: $\alpha_{e,i}^{(2)} \leftarrow a_{e,i}^{(2)} + \epsilon_e^{(2)} \cdot K_{e,i}$,
56 :     $\beta_{e,i}^{(2)} \leftarrow b_{e,i}^{(2)} + z_{e,i}^{(1)} + \sum_j \lambda_e^{(j)} J_e^{(j)} r_{e,i}^{(j)}$.
57 :   endfor
58 :   Compute values $\alpha_e^{(2)} \leftarrow \sum_i \alpha_{e,i}^{(2)}$ and $\beta_e^{(2)} \leftarrow \sum_i \beta_{e,i}^{(2)}$.
59 :   for $i \in [N]$ do
60 :     Compute $\gamma_{e,i}^{(2)} \leftarrow \epsilon_e^{(2)} z_{e,i}^{(2)} - c_{e,i}^{(2)} + \alpha_e^{(2)} b_{e,i}^{(2)} + \beta_e^{(2)} a_{e,i}^{(2)}$.
61 :     if $i = 1$ then set $\gamma_{e,i}^{(2)} \leftarrow \gamma_{e,i}^{(2)} - \alpha_e^{(2)} \cdot \beta_e^{(2)}$.
62 :   endfor
63 :
```

Figure 2.14: blLegRoast.Sign, Part 3 of 4.

```
blLegRoast.Sign$(msg, pk, sk, \tau)$, Part 4

      // Views for final multiplication gate
64 :    for $i \in [N]$ do
65 :       Compute shares: $\alpha_{e,i}^{(3)} \leftarrow a_{e,i}^{(3)} + \epsilon_e^{(3)} \cdot T_{e,i}$
66 :       $\beta_{e,i}^{(3)} \leftarrow b_{e,i}^{(3)} + \sum_j \lambda_e^{(j)} I_e^{(j)} r_{e,i}^{(j)}$
67 :    endfor
68 :    Compute values $\alpha_e^{(3)} \leftarrow \sum_i \alpha_{e,i}^{(3)},\ \ \beta_e^{(3)} \leftarrow \sum_i \beta_{e,i}^{(3)}$.
69 :    for $i \in [N]$ do
70 :       Compute $\gamma_{e,i}^{(3)} \leftarrow \epsilon_e^{(3)} z_{e,i}^{(3)} - c_{e,i}^{(3)} + \alpha_e^{(3)} b_{e,i}^{(3)} + \beta_e^{(3)} a_{e,i}^{(3)}$.
71 :       if $i = 1$ then set $\gamma_{e,i}^{(3)} \leftarrow \gamma_{e,i}^{(3)} - \alpha_e^{(3)} \cdot \beta_e^{(3)}$ endif
72 :       Compute output values: $\omega_{e,i} \leftarrow z_{e,i}^{(2)} + z_{e,i}^{(3)} + \sum_j \lambda_e^{(j)} I_e^{(j)} J_e^{(j)} r_{e,i}^{(j)}$
73 :       if $i = 1$ then set $\omega_{e,i} \leftarrow \omega_{e,i} - \sum_j \lambda_e^{(j)} o_e^{(j)}$ endif
74 :    endfor
75 : endfor
76 : Set $\sigma_4 \leftarrow \left((\alpha_e^{(k)}, \beta_e^{(k)}, (\alpha_{e,i}^{(k)}, \beta_{e,i}^{(k)}, \gamma_{e,i}^{(k)})_{i\in[N]})_{k\in[3]}, (\omega_{e,i})_{i\in[N]}\right)_{e\in[M]}$.
      // Challenger Part 4: Challenge on the sacrificing protocol
77 : Compute challenge hash $h_4 \leftarrow \mathcal{H}_4(h_3, \sigma_4)$
78 : Expand hash $(\bar{i}_e)_{e\in[M]} \leftarrow \mathsf{Expand}(h_4)$, where each $\bar{i}_e \in [N]$.
      // Prover Part 5: Opening the views of the sacrificing protocol
79 : for $e \in [M]$ do
80 :    Set $\mathsf{seeds}_e \leftarrow \{\log_2(N)$ nodes needed to compute $sd_{e,i}$ for $i \in [N] \setminus \{\bar{i}_e\}\}$.
81 : endfor
82 : return $\sigma = \left\{ \begin{array}{c} salt, h_1, h_4, \\ \left(\Delta K_e, \Delta T_e, o_e^{(1)}, \ldots, o_e^{(B)}\right)_{e\in[M]} \\ \left(\alpha_e^{(k)}, \beta_e^{(k)}, \gamma_e^{(k)}, \Delta z_e^{(k)}\right)_{e\in[M],k\in[3]} \\ \left(\mathsf{seeds}_e, \mathsf{com}_{e,\bar{i}}\right)_{e\in[M]} \end{array} \right\}$
```

Figure 2.15: $\mathsf{blLegRoast.Sign}$, Part 4 of 4.

```
$\mathsf{blLegRoast.Verify}(msg, pk_\tau, \sigma)$, Part 1

1 :  Parse signature $\sigma$ as
2 :  $\sigma = \left\{ \begin{array}{c} salt, h_1, h_4, \\ \left(\Delta K_e, \Delta T_e, o_e^{(1)}, \dots, o_e^{(B)}\right)_{e\in[M]} \\ \left(\alpha_e^{(k)}, \beta_e^{(k)}, \gamma_e^{(k)}, \Delta z_e^{(k)}\right)_{e\in[M],k\in[3]} \\ \left(\mathsf{seeds}_e, \mathsf{com}_{e,\bar{i}}\right)_{e\in[M]} \end{array} \right\}$
3 :  Compute $h_2 \leftarrow \mathcal{H}_2\left(h_1, (o_e^{(j)})_{e\in[M],j\in[B]}\right)$.
4 :  Compute $h_3 \leftarrow \mathcal{H}_3\left(h_2, (\Delta z_e^{(k)})_{e\in[M],k\in[3]}\right)$.
5 :  Expand challenge hash 1: $(I_e^{(j)}, J_e^{(j)})_{e\in[M],j\in[B]} \leftarrow \mathsf{Expand}(h_1)$.
6 :  Expand challenge hash 2: $(\lambda_e^{(1)}, \lambda_e^{(2)}, \dots, \lambda_e^{(B)})_{e\in[M]} \leftarrow \mathsf{Expand}(h_2)$.
7 :  Expand challenge hash 3: $(\epsilon_e^{(1)}, \epsilon_e^{(2)}, \epsilon_e^{(3)})_{e\in[M]} \leftarrow \mathsf{Expand}(h_3)$.
8 :  Expand challenge hash 4: $(\bar{i}_e)_{e\in[M]} \leftarrow \mathsf{Expand}(h_4)$.
9 :  for $e \in [M]$ do
10 :   Use $\mathsf{seeds}_e$ to compute $sd_{e,i}$ for $i \in [N] \setminus \{\bar{i}\}$.
11 :   for $i \in [N]$ do
12 :     Sample shares: $\left\{ \begin{array}{c} K_{e,i}, T_{e,i}, \\ r_{e,i}^{(1)}, \dots, r_{e,i}^{(B)}, \\ a_{e,i}^{(1)}, a_{e,i}^{(2)}, a_{e,i}^{(3)}, \\ b_{e,i}^{(1)}, b_{e,i}^{(2)}, b_{e,i}^{(3)}, \\ c_{e,i}^{(1)}, c_{e,i}^{(2)}, c_{e,i}^{(3)}, \\ z_{e,i}^{(1)}, z_{e,i}^{(2)}, z_{e,i}^{(3)} \end{array} \right\} \leftarrow \mathsf{Expand}(sd_{e,i})$.
13 :     if $i = 1$ then
14 :       Adjust shares: $K_{e,1} \leftarrow K_{e,1} + \Delta K_e$,  $T_1 \leftarrow T_1 + \Delta T_e$
15 :       for $k \in [3]$ do
16 :         $c_{e,1}^{(k)} \leftarrow c_{e,1}^{(k)} + \Delta c_e^{(k)}$,  $z_{e,1}^{(k)} \leftarrow z_{e,1}^{(k)} + \Delta z_e^{(k)}$.
17 :       endfor
18 :     endif
19 :     Recompute commitments $\mathsf{com}_{e,i} \leftarrow H_{com}(salt, e, i, sd_{e,i})$.
       // First multiplication gate
20 :     Recompute shares: $\overline{\alpha}_{e,i}^{(1)} \leftarrow a_{e,i}^{(1)} + \epsilon_e^{(1)} \cdot T_{e,i}$,  $\overline{\beta}_{e,i}^{(1)} \leftarrow b_{e,i}^{(1)} + \sum_j \lambda_e^{(j)} r_{e,i}^{(j)}$.
       // Second multiplication gate
21 :     $\overline{\alpha}_{e,i}^{(2)} \leftarrow a_{e,i}^{(2)} + \epsilon_e^{(2)} \cdot K_{e,i}$,  $\overline{\beta}_{e,i}^{(2)} \leftarrow b_{e,i}^{(2)} + z_{e,i}^{(1)} + \sum_j \lambda_e^{(j)} J_e^{(j)} r_{e,i}^{(j)}$.
```

Figure 2.16: $\mathsf{blLegRoast.Verify}$, Part 1.

$\mathsf{blLegRoast.Verify}(msg, pk_\tau, \sigma)$, Part 2

```
        // Final multiplication gates
22 :        $\overline{\alpha}_{e,i}^{(3)} \leftarrow a_{e,i}^{(3)} + \epsilon_e^{(3)} \cdot T_{e,i},\ \overline{\beta}_{e,i}^{(3)} \leftarrow b_{e,i}^{(3)} + \sum_j \lambda_e^{(j)} I_e^{(j)} r_{e,i}^{(j)}.$
23 :        for $k \in [3]$ do
24 :            Recompute share: $\overline{\gamma}_{e,i}^{(k)} \leftarrow \epsilon_e^{(k)} z_{e,i}^{(k)} - c_{e,i}^{(k)} + \alpha_e^{(k)} b_{e,i}^{(k)} + \beta_e^{(k)} a_{e,i}^{(k)}$
25 :            if $i = 1$ then set $\overline{\gamma}_{e,1}^{(k)} \leftarrow \overline{\gamma}_{e,1}^{(k)} - \alpha_e^{(k)} \beta_e^{(k)}$ endif
26 :        endfor
27 :        Recompute output: $\overline{\omega}_{e,i} \leftarrow z_{e,i}^{(2)} + z_{e,i}^{(3)} + \sum_j \lambda^{(j)} I_e^{(j)} J_e^{(j)} r_{e,i}^{(j)}.$
28 :        if $i = 1$ then set $\overline{\omega}_{e,1} \leftarrow \overline{\omega}_{e,1} - \sum_j \lambda_e^{(j)} o_e^{(j)}$ endif
29 :    endfor
30 :    for $k \in [3]$ do
31 :        Compute missing shares: $\overline{\alpha}_{e,\bar{i}}^{(k)} \leftarrow \alpha_e^{(k)} - \sum_{i \neq \bar{i}} \overline{\alpha}_{e,i}^{(k)}$
32 :        $\overline{\beta}_{e,\bar{i}}^{(k)} \leftarrow \beta_e^{(k)} - \sum_{i \neq \bar{i}} \overline{\beta}_{e,i}^{(k)}$
33 :        Compute missing check value share: $\overline{\gamma}_{e,\bar{i}}^{(k)} \leftarrow -\sum_{i \neq \bar{i}} \overline{\gamma}_{e,i}^{(k)}.$
34 :    endfor
35 :    Recompute missing output share: $\overline{\omega}_{e,\bar{i}} \leftarrow -\sum_{i \neq \bar{i}} \overline{\omega}_{e,i}.$
36 :    for $j \in [B]$ do
37 :        Recompute residuosity symbols: $\overline{s}_e^{(j)} \leftarrow \mathcal{L}_0^\ell(o_e^{(j)}) - \mathsf{pk}_{I_e^{(j)}}.$
38 :    endfor
39 : endfor
40 : Check 1: $h_1 = H_1(msg, salt, ((\mathsf{com}_{e,i})_{i \in [N]}, (\overline{s}_e^{(j)})_{j \in [B]}, \Delta K_e, \Delta T_e, (\Delta c_e^{(k)})_{k \in [3]})_{e \in [M]})$
41 : Check 2: $h_4 = H_4\left(h_3, \left((\alpha_e^{(k)}, \beta_e^{(k)}, (\overline{\alpha}_{e,i}^{(k)}, \overline{\beta}_{e,i}^{(k)}, \overline{\gamma}_{e,i}^{(k)})_{i \in [N]})_{k \in [3]} (\omega_{e,i})_{i \in [N]}\right)_{e \in [M]}\right)$
42 : Output accept if both checks pass, fail otherwise.
```

Figure 2.17: $\mathsf{blLegRoast.Verify}$, Part 2.

2.7.2 blLegRoast Proof

We now show that blLegRoast satisfies the independent blinding property (7). We define randBlind by simply using a random $T \in \mathbb{F}_p^\times$ instead of having it determined by the hash function. To show that $\mathbf{Adv}^{\mathsf{Ind-Blind}}(\mathcal{A})$ is small, we need to reduce to the security of $\mathcal{L}$ the ability to distinguish whether $K_1 = K_2 = \cdots = K_n$ or not in the following matrix:

$$\begin{bmatrix} F_{\mathcal{I}}^{\ell}(K_1) + F_{\mathcal{J}}^{\ell}(T_1) \\ F_{\mathcal{I}}^{\ell}(K_2) + F_{\mathcal{J}}^{\ell}(T_2) \\ \vdots \\ F_{\mathcal{I}}^{\ell}(K_n) + F_{\mathcal{J}}^{\ell}(T_n) \end{bmatrix} = \begin{bmatrix} \mathcal{L}_{K_1}^{\ell}(i_1) + \mathcal{L}_{T_1}^{\ell}(j_1) & \dots & \mathcal{L}_{K_1}^{\ell}(i_L) + \mathcal{L}_{T_1}^{\ell}(j_L) \\ \mathcal{L}_{K_2}^{\ell}(i_1) + \mathcal{L}_{T_2}^{\ell}(j_1) & \dots & \mathcal{L}_{K_2}^{\ell}(i_L) + \mathcal{L}_{T_2}^{\ell}(j_L) \\ \vdots & \ddots & \vdots \\ \mathcal{L}_{K_n}^{\ell}(i_1) + \mathcal{L}_{T_n}^{\ell}(j_1) & \dots & \mathcal{L}_{K_n}^{\ell}(i_L) + \mathcal{L}_{T_n}^{\ell}(j_L) \end{bmatrix}. \tag{2.3}$$

The natural property of $\mathcal{L}$ to reduce to is its security as a PRF. If, for each key T_i the result on the inputs $\{j_m\}$ for $m \in [L]$ is indistinguishable from random, then each row of the matrix is indistinguishable from uniformly random, independent of whether the K_i values are all the same or not. This means that rather than relying on a search version of the problem that defines the security of the Legendre PRF, we require a decisional version.

Definition 10 (Decisional Fixed Input Power Residue Symbol Problem)**.** Let p be an odd prime and ℓ be a positive integer with $\ell \mid p-1$. Let $\mathcal{J} = (j_1, j_2, \dots, j_L) \in \mathbb{F}_p^L$. Let $\mathcal{O}_{\mathsf{Pow}}$ be an oracle that, when queried samples a random $T \leftarrow_{\$} \mathbb{F}_p$ and returns $F_{\mathcal{J}}^{\ell}(T)$. Let $\mathcal{O}_{\mathsf{Ran}}$ be an oracle that returns a uniform element form $\mathbb{Z}_\ell^L$. The *Decisional Fixed Input Power Residue Symbol Problem* is, given $p, \ell, \mathcal{J}, L$ as input distinguish the two oracles with non-negligible probability. We write $\mathbf{Adv}^{\mathsf{Ind-PRF}}(\mathcal{A})$ to denote

$$\left| \Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Pow}}}] - \Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Ran}}}] \right|.$$

We note that the security of "plain" LegRoast does not depend on such a property. It requires only a search version of the problem, where $\mathcal{A}$ is required to actually find the corresponding T value. The security of the decisional variant and its relation to the search variant is still conjectural. Damgård's original 1988 paper [58] that proposed the use of the Legendre function as a PRF investigates statistical properties with respect to standard randomness tests, but this does not constitute a serious cryptanalytic effort. Grassi et al.'s 2016 paper [82] is largely responsible for the renewed interest in the Legendre function as a PRF. They defined a decisional variant similar to our own, except allowing for adaptive queries. However, later cryptanalytic works [102, 33] focused on the search variant. Security of our scheme relies on the decisional variant; significantly more research on the hardness of the decisional variant is needed before this scheme can be confidently used.

Lemma 9. *If the Legendre PRF is pseudo-random, then* $\mathsf{blLegRoast}$ *satisfies the independent blinding property (7). In particular, for any adversary* $\mathcal{A}$ *that distinguishes a random oracle from the power residue symbols with advantage* $\mathbf{Adv}^{\mathsf{Ind-PRF}}(\mathcal{A})$ *(defined above), we have that* $\mathbf{Adv}^{\mathsf{Ind-Blind}}_{\mathsf{blLegRoast}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathsf{Ind-PRF}}(\mathcal{A})$.

Proof. Let game G_0 be one in which $\mathcal{A}$ is given independent blindings of the same public key, and game G_3 be where $\mathcal{A}$ is given independent blindings of independent public keys. We will introduce intermediate games G_1 and G_2, and bound the difference in probability between subsequent hops.

In game G_1, we replace the correct values from $F^{\ell}_{\mathcal{J}}(T_i)$ with uniformly random outputs in $\mathbb{Z}^L_{\ell}$ for $i \in [n]$. Since the T_i values are uniformly random already, this replacement causes at most a difference $\mathbf{Adv}^{\mathsf{Ind-PRF}}(\mathcal{A})$, that is $|\Pr[1 \leftarrow \mathcal{A}^{G_0}] - \Pr[1 \leftarrow \mathcal{A}^{G_1}]| \leq \mathbf{Adv}^{\mathsf{Ind-PRF}}$.

As $F^{\ell}_{\mathcal{I}}(K)$ is now being added to uniformly random values, we can replace it with whatever we like and cause no change in the distribution. Thus, for game G_2 we replace each the n instances of $F^{\ell}_{\mathcal{I}}(K)$ by drawing a random K_i and using $F^{\ell}_{\mathcal{I}}(K_i)$ instead. Again, this does not change the distribution of what the adversary has access to, so $|\Pr_{G_1}[1 \leftarrow \mathcal{A}] - \Pr_{G_2}[1 \leftarrow \mathcal{A}]| = 0$.

Finally, for game G_3, we swap the $F^{\ell}_{\mathcal{J}}(T_i)$ values *back* to being honestly generated from T_i, instead of just uniformly random. At this point we have correctly generated $F^{\ell}_{\mathcal{I}}(K_i)$ and $F^{\ell}_{\mathcal{J}}(T_i)$ values, with all K_i uniform and independent. This again introduces a difference bounded by $\mathbf{Adv}^{\mathsf{Ind-PRF}}$. Summing up the differences across all games, we obtain

$$\left|\Pr_{G_0}[1 \leftarrow \mathcal{A}] - \Pr_{G_3}[1 \leftarrow \mathcal{A}]\right| \leq 2 \cdot \mathbf{Adv}^{\mathsf{Ind-PRF}}(\mathcal{A}).$$

□

It remains to be shown that the signing with oracle reprogramming property is satisfied and the the scheme is existentially unforgeable. In LegRoast, the proof of existential unforgeability has a fairly straightforward structure. First, the authors show that an adversary $\mathcal{A}$ that attacks the existential unforgeability under chosen message attack can be used to construct an adversary $\mathcal{R}$ that makes no signing queries. This is done by essentially proving the signing with oracle reprogramming property that we defined in Section 2.4. Then to justify the key-only security, they show that with high probability, an adversary that has successfully constructed a signature has, with all but negligible probability, queried a witness to a β-relaxation of the PRF relation. Finally, they show that, with all but negligible probability, the only witness to such a relaxation is the actual secret key K.

We take a similar approach for the blinded version of LegRoast. The proof that an EU-CMA adversary implies a EU-KO one also works as a proof of the signing with oracle reprogramming property, and closely mirrors the original proof. The proof showing that a key only adversary can be used to recover a witness for the relaxed relation is also similar, with appropriate modifications made for the additional multiplication gates and the extra round. Finally, the proof that the only witness of the relaxation is the original (K, T) proceeds similarly, with a small modification needed to account for the flexibility in having the T parameter. We will present these proofs in the opposite order, starting by talking about the underlying relation and building towards the proof of the EU-CMA security.

Underlying (relaxed) relation

The security of the blinded LegRoast scheme depends on the inability of the adversary to find values K and T such that $F^{\ell}_{\mathcal{I}}(K) + F^{\ell}_{\mathcal{J}}(T) = \mathit{blpk}$. In the signature scheme, because only a random subset of the indices of the public key are verified, we need to consider a β-relaxation of finding such a K and T, where only 'enough' of the indices match. We adapt Definitions 1 and 2 from [34] to our situation.

Definition 11 (Additive ℓth-power residue PRF relation)**.** For an odd prime p, a positive integer $\ell \mid p-1$ and lists $\mathcal{I}, \mathcal{J} \in \mathbb{Z}_p^L$ we define the additive ℓth power residue PRF relation $R^+_{\mathcal{L}^\ell}$ with output length L as

$$R^+_{\mathcal{L}^\ell} = \{(F^{\ell}_{\mathcal{I}}(K) + F^{\ell}_{\mathcal{J}}(T), (K, T)) \in \mathbb{Z}_\ell^L \times \mathbb{F}_p \times \mathbb{F}_p\}.$$

Definition 12 (Additive β-approximate PRF relation)**.** For $\beta \in [0, 1]$, an odd prime p, a positive integer $\ell \mid p-1$, and lists $\mathcal{I}, \mathcal{J} \in \mathbb{Z}_p^L$ define the additive β-approximate PRF relation $R^+_{\beta\mathcal{L}^\ell}$ with output length L as

$$R^+_{\beta\mathcal{L}^\ell} = \{(s, (K, T)) \in \mathbb{Z}_\ell^L \times \mathbb{F}_p \times \mathbb{F}_p \mid \exists a \in \mathbb{Z}_\ell : d(s + (a, \ldots, a), F^{\ell}_{\mathcal{I}}(K) + F^{\ell}_{\mathcal{J}}(T)) \leq \beta L\},$$

where $d(\cdot, \cdot)$ denotes the Hamming distance.

We can then adapt Theorem 1 from [34] to our situation.

Theorem 3. *Let $\mathcal{B}(n, q)$ denote the binomial distribution with n samples each with success probability q. Let $\mathcal{C}(n, q, m)$ denote the cumulative distribution function with at least m successes, i.e., $\mathcal{C}(n, q, m) = \Pr[B(n, q) \geq m]$. Take $K, T \in \mathbb{F}_p$ and $s = F^{\ell}_{\mathcal{I}}(K) + F^{\ell}_{\mathcal{J}}(T)$. Then with probability at least*

$$1 - \ell p^2 \cdot \mathcal{C}\left(L, \frac{1}{\ell} + \frac{1}{\sqrt{p}} + \frac{2}{p}, (1-\beta)L\right)$$

over the choice of $\mathcal{I}, \mathcal{J}$, there exists only one witness for $s \in R^+_{\beta\mathcal{L}^\ell}$, which is (K, T), the witness for the exact relation $R^+_{\mathcal{L}^\ell}$.

To prove the theorem we will require Lemma 1 from [34].

Lemma 10. *Let p be a prime and $\ell \mid p-1$. For any $K, K' \in \mathbb{F}_p$ with $K \neq K'$, and $a \in \mathbb{Z}_\ell$, we have*

$$\Pr_{i \leftarrow\$ \mathbb{F}_p}[\mathcal{L}^\ell(K+i) = \mathcal{L}^\ell(K'+i) + a] \leq \frac{1}{\ell} + \frac{1}{\sqrt{p}} + \frac{2}{p}.$$

Proof of Theorem 3. For any $K', T', j \in \mathbb{F}_p$ with $K' \neq K$, and any $a \in \mathbb{Z}_\ell$, we let $a' = \mathcal{L}^\ell(T'+j) - \mathcal{L}^\ell(T+j) + a$. Then by Lemma 10 we have that the probability, over the choice i that $\mathcal{L}^\ell(K+i) = \mathcal{L}^(K'+i) + a'$ is bounded. Rearranging the terms of a' we have a bound on the probability that $\mathcal{L}^\ell(K+i) + \mathcal{L}(T+j) = \mathcal{L}^\ell(K'+i) + \mathcal{L}^\ell(T'+j) + a$. As each of the i values is sampled independently, we get that the probability that for a tuple (K', T', a) we have that $d(F^\ell_\mathcal{I}(K') + F^\ell_\mathcal{J}(T'), F^\ell_\mathcal{I}(K) + F^\ell_\mathcal{J}(T) + (a, \ldots, a)) \leq \beta L$ is

$$C(L, 1/\ell + 1/\sqrt{p} + 2/p, (1-\beta)L).$$

This is true for any $K' \neq K$, and any T' or a. There are $(p-1)$ choices for K', p choices for T', and ℓ choices for a. So the probability that there exists such a K', T', and a is upper bounded by the previous probability multiplied by $\ell(p-1)p$, which we replace with ℓp^2 for simplicity. □

Relaxed relation implies key-only security

Theorem 4. *Let q_{com}, q_1, q_2, q_3, and q_4 be the number of queries that an adversary $\mathcal{A}$ makes to random oracles H_{com}, H_1, H_2, H_3, and H_4 respectively. Fix a constant $\beta \in \{0, 1\}$. If $\mathcal{A}$ succeeds in breaking the existential unforgeability of blLegRoast with advantage a then there exists an adversary $\mathcal{R}$ capable of finding a β-approximate witness for blpk with probability at least*

$$a - \frac{MN(q_{sd} + q_1 + q_2 + q_3 + q_4)^2}{2^{2\lambda}} - \Pr[X + Y + Z + W = M]$$

where X is a r.v. distributed as the $\max(X_1, \ldots, X_{q_1})$, with each X_i distributed as $\mathcal{B}(M, (1-\beta)^B)$, and Y, Z, and W defined similarly, but with:

- *Y_i ranging from $i = 1$ to q_2 and Y_i distributed as $\mathcal{B}(M - X, 1/p)$,*

- *Z_i ranging from $i = 1$ to q_3 and Z_i distributed as $\mathcal{B}(M - X - Y, 3/p)$,*
- *W_i ranging from $i = 1$ to q_4 and W_i distributed as $\mathcal{B}(M - X - Y - Z, 1/N)$.*

Proof. We define how the reduction algorithm $\mathcal{R}$ will operate. To begin with, $\mathcal{R}$ is provided $s = F^{\ell}_{\mathcal{I}}(K) + F^{\ell}_{\mathcal{J}}(T)$ which is passed along to $\mathcal{A}$ as the blinded public key. $\mathcal{R}$ will maintain $H_{com}, H_1 H_2, H_3$, and H_4 as random oracles. Typical to random oracle proofs, $\mathcal{R}$ will employ the 'lazy sampling' methodology, and respond consistently when given repeated queries and otherwise sample a random output and provide it to $\mathcal{A}$, maintaining a table of all inputs and outputs. In general, the sampled output is entirely uniform, with some small exceptions:

- The same value is never returned twice (no collisions).
- When the adversary makes a query to H_1 of the form $\big((\mathsf{com}_{e,i})_{i\in[N]}, \dots\big)_{e\in[M]}$, add the $\mathsf{com}_{e,i}$ values to a list of values to never return for new queries. This means that $\mathcal{A}$ cannot cannot provide a commitment and then later find an opening to that commitment.
- Do the same thing for queries to H_2 of the form (h_1, σ_2) (do not return h_1), queries to H_3 of the form (h_2, σ_3) (do not return h_2), and queries to H_4 of the form (h_3, σ_4) (do not return h_3). This ensures that the adversary is forced to adhere to the 'correct' order of the Fiat–Shamir paradigm.

Note that $\mathcal{R}$ clearly runs in time roughly the same as $\mathcal{A}$. We will show that, if $\mathcal{A}$ succeeds then with high probability, embedded into $\mathcal{A}$'s queries to the random oracles is enough information to recover a witness (K', T').

The remainder of the proof works very similarly as in [34]. We call specific attention to the cases where the proof differs.

Extracting a β-relaxed witness. To recover a witness, $\mathcal{R}$ looks at the queries made to the hash function. Let $\mathcal{Q}_i$ denote the set of query-responses to oracle H_i. For each query $\sigma_1 = ((\mathsf{com}_{e,i})_{i\in[N]}, \dots, \Delta K_e, \Delta T_e, \dots)_{e\in[M]}$ to H_1, check and see if each $\mathsf{com}_{e,i}$ is the output for a query $(salt, e, i, sd_{e,i})$ to H_{com}. If so, then by expanding $sd_{e,i}$ to recover $K_{e,i}$ for each e, i, summing them together over i and adding in ΔK_e, then doing the same for T, we have a candidate witness. For each query to H_1 where this is possible, we maintain a table $\mathcal{T}_i$ of inputs. The table $\mathcal{T}_i$ is indexed by a query σ_1 and the round e and contains the values $K_e, T_e, (r^{(j)}_{e,i})_{i\in[N],j\in[B]}, (a^{(k)}_{e,i}, b^{(k)}_{e,i}, c^{(k)}_{e,i}, z^{(k)}_{e,i})_{i\in[N],k\in[3]}$. Our task is to show that if no such candidate witness can be constructed, then it is only with negligible probability that $\mathcal{A}$ can create a valid signature.

Cheating in the First Phase. Let $\sigma_{best_1}, h_{best_1}$ be the best query-response pair from H_1 that $\mathcal{A}$ receives. 'Best' here refers to maximizing the probability that $\mathcal{A}$ can cheat from this query because the chosen indices that result from the output 'line up' in a way favourable to the adversary. For a query-response pair $((msg, salt, \sigma_1), h_1)$ to H_1, define $G_1(\sigma_1, h_1 = \{I_e^{(j)}\}_{e\in[M],j\in[B]})$ as the rounds $e \in [M]$ for which:

- An entry at index $[\sigma_1, e]$ exist in $\mathcal{T}_i$.
- $\mathcal{L}^\ell((K_e + I_e^{(j)})(T_e + J_e^{(j)})r_e^{(j)}) = s_e^{(j)} + pk_{I_e^{(j)},J_e^{(j)}}$ for all $j \in [B]$.

We then rely on the following lemma from [34]:

Lemma 11. *If a witness (K, T) cannot be recovered from the queries to H_1 then for any positive integer x, we have that for all query-response pairs to H_1*

$$\Pr[\#G_1(\sigma_1, e) > x] \leq \Pr[X > x]$$

where X is a random variable distributed as $\max(X_1, X_2, \ldots, X_{q_1})$ *where each X_i is i.i.d. $\mathcal{B}(M, (1-\beta)^B)$.*

Cheating in the second phase. For the second round, any 'functional' (could possibly verify) query-response pair $((h_1, \sigma_2 = (o_e^{(j)})_{e\in[M],j\in[B]}), (\lambda_e^{(j)})_{e\in[M],j\in[B]})$ we define the set of good rounds $G(h_1, \sigma_2, h_2)$ as:

- $\emptyset$ if h_1 is not the output of a query σ_1 to H_1. Otherwise this cannot lead to a valid signature. Let $\sigma_1 = (\ldots, (s_e^{(j)})_{j\in[B]}, \ldots)$.
- $\emptyset$ if there is an index (e, j) such that $\mathcal{L}^\ell(o_e^{(j)}) \neq s_e^{(j)} + pk_{I_e^{(j)},J_e^{(j)}}$. If there is, then the signature will not verify, because the check on h_1 will fail when reconstructing the s values.
- The values $e \in [M]$ for which the following procedure passes:
  - Using the associated σ_1, recover the inputs from $\mathcal{T}[\sigma_1, e]$.

– With these inputs, as well as the $I_e^{(j)}$, $J^{(e)}$, $o_e^{(j)}$, and $\lambda_e^{(j)}$ values, calculate if the error term E found in Equation 2.2 is equal to 0.

We once again must establish that if a witness for the relation cannot be recovered from the inputs, then the size of $G_2(h_1, \sigma_2, h_2)$ is bounded. Obviously we must be in the third case, or else $G_2(h_1, \sigma_2, h_2)$ is empty. There are two possibilities:

- A given index $e \in [M]$ is also in $G_1(\sigma_1, h_1)$.
- If the index in not in $G_1(\sigma_1, h_1)$, then it must be due to the fact that there is an index $j \in [B]$ such that $\mathcal{L}^\ell((K_e + I_e^{(j)})(T_e + J_e^{(j)})r_e^{(j)}) \neq s_e^{(j)} + pk_{I_e^{(j)}, J_e^{(j)}}$. But since we have that $\mathcal{L}^\ell(o_e^{(j)}) = s_e^{(j)} + pk_{I_e^{(j)}, J_e^{(j)}}$, this means that the same index has the property that $\mathcal{L}^\ell((K_e + I_e^{(j)})(T_e + J_e^{(j)})r_e^{(j)}) \neq o_e^{(j)}$. This in turn means that the error term is a non-zero linear function in the $\lambda_e^{(j)}$ values. So, over the random choices of λ, the probability that the error term works out to be 0 anyways is equal to $1/p$. This allows us to establish the following Lemma:

Lemma 12. *If a witness (K, T) cannot be recovered from the queries to H_1, then for any positive integer x, we have that for all query-response pairs to H_2*

$$\Pr[\#G_2(h_1, \sigma_2, h_2) > x] leq \Pr[X + Y > x]$$

where X is a random variable distributed as in Lemma 11, and Y is distributed as $\max(Y_1, Y_2, \ldots, Y_{q_2})$ with each Y_i is i.i.d. $\mathcal{B}(M - X, \frac{1}{p})$.

This proof essentially states that we must either be in the first or second case, and that in the second case the probability that a round works out is bounded by $1/p$, as already discussed. Remaining details for this part of the proof can similarly be found in [34].

Cheating in the third phase. For round three, we continue with our characterization of the best possible input an adversary can construct. Recall that the input to H_3 consists of an h_2 and the $\Delta z_e^{(k)}$ values. For an input-output pair, define the set $G_3(h_2, \sigma_3, h_3)$ as

- $\emptyset$ if h_2 is not the output of a previous query (h_1, σ_2) to H_2, which in turn is associated with a query σ_1 that has already been made to H_1. Otherwise, the signature will never verify.

- The values $e \in [M]$ such that the following procedure passes: trace back to the input to the fist phase and recover the seeds in order to generate the shares of K, T, $r^{(j)}$, $a^{(k)}$, $b^{(k)}$, $c^{(k)}$, and $z^{(k)}$. By summing over these shares and incorporating the committed to Δ values, we get the candidate state of the equations at the time when the $\epsilon^{(k)}$ values become defined. This in turn allows us to define the corresponding $\alpha^{(k)}$, $\beta^{(k)}$, $\gamma^{(k)}$, and $\omega^{(k)}$ values, by

$$\begin{aligned}
\alpha_e^{(1)} &= a_e^{(1)} + \epsilon_e^{(1)} \cdot T_e & \beta_e^{(1)} &= b_e^{(1)} + \sum_j \lambda_e^{(j)} r_e^{(j)} \\
\alpha_e^{(2)} &= a_e^{(2)} + \epsilon_e^{(2)} \cdot K_e & \beta_e^{(2)} &= b_e^{(2)} + z_e^{(1)} + \sum_j \lambda_e^{(j)} J_e^{(j)} r_e^{(j)} \\
\alpha_e^{(3)} &= a_e^{(3)} + \epsilon_e^{(3)} \cdot T_e & \beta_e^{(3)} &= b_e^{(3)} + \sum_j \lambda_e^{(j)} I_e^{(j)} r_e^{(j)} \\
\gamma_e^{(k)} &= \epsilon_e^{(k)} z_e^{(k)} - c_e^{(k)} + \alpha_e^{(k)} b_e^{(k)} + \beta_e^{(k)} a_e^{(k)} - \alpha_e^{(k)} \beta_e^{(k)} \\
\omega_e &= z_e^{(2)} + z_e^{(3)} + \sum_j \lambda_e^{(j)} I_e^{(j)} J_e^{(j)} r_e^{(j)} - \sum_j \lambda_e^{(j)} o_e^{(j)}.
\end{aligned}$$

  Then pass if all of $\gamma_e^{(k)}$ and $\omega_e^{(k)}$ are equal to zero.

Again, we need to bound the number of such rounds in the event that a witness cannot be recovered from the queries to H_1. Clearly it must be the case that we can trace the query h_2 back to the original input to H_1, or else the size of $G_3(h_2, \sigma_3, h_3)$ is zero. For each index $e \in [M]$, it may be the case that the index is in $G_2(h_1, \sigma_2, h_2)$. Assume it is not. Since the index is not in G_2, we can see that it must be the case that the error term is not equal to zero. A simple reduction shows that if $x^{(k)}$ and $y^{(k)}$ are the inputs to the kth multiplication gate, we have that,

$$\gamma_e^{(k)} = \epsilon_e^{(k)} \cdot (z_e^{(k)} - x_e^{(k)} \cdot y_e^{(k)}) - c_e^{(k)} + a_e^{(k)} \cdot b_e^{(k)}.$$

We can see that this is equal to zero only if either:

- $z_e^{(k)} = x_e^{(k)} \cdot y_e^{(k)}$ and $c_e^{(k)} = a_e^{(k)} \cdot b_e^{(k)}$ (that is, the $\Delta z_e^{(k)}$ and $\Delta c_e^{(k)}$ values were chosen properly so that z and c really are equal to the product of the inputs).
- $\epsilon_e^{(k)} = (a_e^{(k)} \cdot b_e^{(k)} - c_e^{(k)})(z_e^{(k)} - x_e^{(k)} \cdot y_e^{(k)})^{-1}$.

As the $\epsilon^{(k)}$ values are chosen uniformly at random, for each of the γ values, the chances that $\gamma_e^{(k)} = 0$ *without* $z_e^{(k)} = x_e^{(k)} \cdot y_e^{(k)}$ is $1/p$. But if, for each k, we do have $z_e^{(k)} = x_e^{(k)} \cdot y_e^{(k)}$

then the corresponding ω_e value will in fact be equal to the error term. But since we are assuming that this round is not in $G_2(h_1, \sigma_2, h_2)$ we have that the error term is in fact not equal to zero. So we have that for least one of the $\gamma_e^{(k)}$ terms we must have that $z_e^{(k)}$ is not equal to the inputs. Thus the probability that all values are equal to zero is at most $3/p$.

This allows us, as in the previous round, to bound the size of $G_3(h_2, \sigma_3, h_3)$:

Lemma 13. *If a witness (K, T) cannot be recovered from the queries to H_1, then for any positive integer x, we have that for all query-response pairs to H_3*

$$\Pr[\#G_3(h_2, \sigma_3, h_3) > x] \leq \Pr[X + Y + Z > x]$$

where X is a random variable distributed as in Lemma 11, and Y is distributed as in 12, and Z is distributed as $\max(Z_1, Z_2, \ldots, Z_{q_2})$ *with each Z_i is i.i.d. $\mathcal{B}(M - X - Y, \frac{3}{p})$.*

Cheating in the fourth phase. Assume without loss of generality that $\mathcal{A}$ verifies the signature that they submit as a forgery. Recall that queries to H_4 should take the form

$$h_3, \sigma_4 = \left((\alpha_e^{(k)}, \beta_e^{(k)}, (\alpha_{e,i}^{(k)}, \beta_{e,i}^{(k)}, \gamma_{e,i}^{(k)})_{i \in [N]})_{k \in [3]}, (\omega_{e,i})_{i \in [N]}\right)_{e \in [M]}.$$

For each such query σ_4 we can bound the probability that the response leads $\mathcal{A}$ to be able to construct a valid signature.

As usual, we note that h_3 must be the output of a query to H_3 — if the adversary decided to come up with the h_3 value in some other way, then after they have queried it to H_4 no input that they provide to H_3 will lead to h_3. Similarly, we can work our way backwards to the original query to H_1.

Assume that in a given round e we have that $e \notin G_3(h_2, \sigma_3, h_3)$. Then it is the case that at least one of the associated $\gamma_e^{(k)}$ or ω_e values is not equal to zero. In this case, the only way that we can end up with a valid signature is if exactly $N - 1$ of the parties behave honestly. If all parties behave honestly, then the final signature will not validate because the $\gamma_{e,\bar{i}_e}^{(k)}$ or $\omega_{e,\bar{i}_e}$ values will not be correct. If more than $N - 2$ parties misbehave then the verifier will not replicate that misbehaviour. With exactly $N - 1$ parties misbehaving, then for any round, the probability that that party is chosen to stay concealed by the protocol is clearly $1/N$. As a result, we can give our final bound on the probability that the adversary winning.

The reduction simulates all oracles correctly, except that it does not return collisions or allow the adversary to cheat at commitments to the random oracles. In [34], the authors establish that the probability that an adversary notices this inconsistency is at most

$$\frac{MN(q_{sd}+q_1+q_2+q_3+q_4)^2}{2^{2\lambda}}. \tag{2.4}$$

And given that the adversary does not notice this incongruence we have that the probability that they are able to find a signature is less than

$$\Pr[X+Y+Z+W=M],$$

where X, Y, and Z are distributed according to Lemmas 11, 12, and 13, and W is distributed as $\max(W_1,\ldots,W_{q_4})$ with each W_i distributed i.i.d. as $\mathcal{B}(M-X-Y-Z,1/N)$. $\square$

EU-CMA implies key-only security and signing with oracle reprogramming

In order to simulate signatures with oracle reprogramming, we can define our Forge function as follow.

1. To cheat in the first round, the simulator simply samples the values ΔK_e and ΔT_e uniformly at random, rather than calculating them using T. The simulator also aborts if a salt is reused from an earlier round. The reduction queries the random oracle (which is also managed by them) to obtain h_1, which is expanded to the $I_e^{(j)}, J_e^{(j)}$ values.

2. For the second part of proving, rather than genuinely generating the $o_e^{(j)}$ values, the simulator will sample them uniformly and then post select so that $\mathcal{L}^\ell(o_e^{(j)})-s_e^{(j)}=blpk_{I_e^{(j)},J_e^{(j)}}$ (since this can be done independently for each j and $\mathcal{L}^\ell(o_e^{(j)})$ takes on one of ℓ values this is not hard). Then they again query this to the random oracle to obtain the $\lambda_e^{(j)}$ values.

3. For prover part 3, simply select each $\Delta z_e^{(k)}$ value to be uniformly random. Set σ_3 in the usual way and query for the $\epsilon_e^{(k)}$ values.

4. For the simulation of prover part 4, the simulator will need to cheat so the openings can be provided for the requested parties that look correct, and the overall proof lines up. To do this, they will decide which parties will be opened in advance, and then later program the random oracle to provide this output. So for each $e\in[M]$, the simulator samples $\bar{i}_e\in\{1,\ldots,N\}$ at random. Then it behaves entirely honestly except for when working with the values for the $\bar{i}_e$th party in round e. This party

is calculated last, and we set the γ and ω values according to how they are set in verification, rather than how they are set in an honest signing instance. That is, we set

$$\gamma^{(k)}_{e,\bar{i}_e} \leftarrow -\sum_{i \neq \bar{i}_e} \gamma^{(k)}_{e,i} \qquad\qquad \omega_{e,\bar{i}_e} \leftarrow -\sum_{i \neq \bar{i}_e} \omega_{e,i}.$$

Then σ_4 is prepared accordingly. Rather than querying the random oracle H_4, we instead reprogram it so that (h_3, σ_4) maps to the desired $(\bar{i}_e)_{e \in [M]}$ values.

5. Finally for part 5, the simulator can generate the seeds as desired and return the signature as expected.

Simulation Indistinguishability. The random oracles are all simulated perfectly. The programmed output is chosen uniformly at random, and so as long as it has not been previously queried, the programmed output is indistinguishable from a real one. As the oracle H_4 is reprogrammed on the point (h_3, σ_4), this is the output of the Ext function. That is, Ext takes in the signature σ, uses it to reconstruct h_3 and σ_4 according to the Verify function, and returns them as the reprogrammed point.

The min-entropy of this point is quite high. We need to consider the min-entropy of this input from the beginning of the signing routine. All of the α, β, and γ values are derived in part from the a, b, c, and z values. These in turn are derived be expanding the seeds, which are chosen from the uniformly sampled root seed sd_e. Therefore the entropy of the programmed point is bounded by the entropy of the sampled root vector. As a root vector is sampled for each round, the min entropy of the overall programmed point can be bounded by $2^{\lambda \cdot M}$.

We must consider the distribution of the reprogrammed point and the returned signature. Most of the parts of the signature are exactly the same as what they would have been without any cheating (just possibly calculated a different way), except for the Δ correction values: ΔK_e, ΔT_e, and $\Delta z^{(k)}_e$ are all chosen uniformly at random rather than calculated based on K, T, and the output values. But, this does not actually change the overall distribution of the signatures. Since the unknown shares $K_{e,\bar{i}_e}$, $T_{e,\bar{i}_e}$, $z^{(k)}_{e,\bar{i}_e}$ are never seen by verifier, and are generated uniformly from $\mathsf{Expand}(\mathsf{sd}_{e,\bar{i}_e})$, they just as easily could have been the values such that the Δ values are calculated correctly. Therefore the distribution, conditioned on what a verifier sees as part of the signature, is exactly the same, and we can sign with oracle reprogramming.

We are being slightly duplicitous in the above statements about indistinguishability. In general, the output of Expand has been considered to be a collection of elements in $\mathbb{F}_p$, but in actuality it would be instantiated by a pseudo-random generator, and thus, be a string of bits. The reason we are lax about distinguishing these two sets is because in practice, we can choose our parameters so that the two are nearly identical. LegRoast takes p as the Mersenne prime $2^{127} - 1$, and instantiates Expand by taking the output of a PRNG sixteen bytes at a time and interpreting the result as an integer and reducing modulo p. So, if we are given a uniform element x in $\mathbb{F}_p$ and want to convert this into a nearly-uniform bit string, we can simply sample a bit b and take the bit-representation of $x + b \cdot (2^{127} - 1)$. This distribution is identical, except that we cannot get the all-one bit string with this method. As this bit string is only ever part of the output of Expand with negligible probability, we need not worry.

The distribution of the programmed output, y_{forged} is entirely unchanged from y_{real}, as both are taken uniformly from $\{1, \ldots, N\}^M$, and independent of anything else in the signatures. So we can conclude that the statistical distance δ between our real and forged σ and y is in fact 0.

For the complete details on how signing with oracle reprogramming shows that an adversary attacking the existential unforgeability can be used to construct an adversary attacking the key-only security, we refer to the LegRoast paper [34], which shows exactly this in their Lemma 3.

2.8 A Generic Zero-Knowledge Key-Blinding Scheme

In this section we explain how one of the most unique submissions to NIST's post-quantum standardization effort, Picnic [50], can also be transformed into a key-blinded signature scheme. Picnic improves upon techniques introduced in [93, 80] to construct a signature scheme out of the generic zero-knowledge proof system ZKB++. A secret key in Picnic is the secret key to a symmetric key encryption function, $k \in \{0, 1\}^\lambda$. The public key is a pair of values $(x, y) \in \{0, 1\}^{2\lambda}$ such that $y = \mathsf{Enc}_k(x)$ for some suitable encryption function Enc.

To sign a message msg, the signer constructs a generic zero-knowledge proof, dependent on msg that they know a key k under which x is encrypted to y. The hash of msg is then mixed into the randomness used to generate this zero-knowledge proof in such a way to result in an existentially unforgeable signature scheme. The zero-knowledge proof itself is based on the 'MPC in the head' methodology proposed in [93], which noted a fundamental connection between zero-knowledge proofs and multi-party computation (MPC).

We present a rough outline of the signature scheme Picnic, and refer to [50] for full details. Consider three separate parties, A, B, and C, who possess private values a, b, and c, respectively. They are using multi-party computation to compute $y = \mathsf{Enc}_k(x)$ for some fixed value x and $k = a \oplus b \oplus c$. In the end, no party will learn k. Indeed, for certain configurations of multi-party computation, even if two out of the three parties collaborate, they cannot learn k.

We can consider the views of each of these parties, consisting of all incoming and outgoing messages as well as all intermediate values in the computation. Because parties can collaborate and still not learn k, the views of two out of the three parties similarly do not contain enough information to decide k. However these views also attest—to a certain extent—to the validity of the computation: One can examine the transcript of the parties views to ascertain that all computations and incoming and outgoing messages were computed correctly.

This realization allows the authors of [93] to show that a secure MPC scheme can be transformed into a zero-knowledge proof. Someone who knows the secret key k can split it up into three parts, a, b, and c, and then run the MPC protocol 'in their head', with each party having a part of k. The respective transcripts of each parties view of the protocol can be committed to, after which a verifier can challenge the prover to reveal the views of two out of the three parties. For a prover to cheat, they must make at least one of the parties misbehave, which can be detected by the verifier with constant probability. But the scheme is actually zero knowledge, as transcripts can easily be simulated by having the party who is not revealed misbehave. By applying the Fiat–Shamir transform, one can obtain a signature scheme: this is Picnic.

Adding a key-blinding functionality to Picnic can be done by encrypting the public key a second time, this time under information derived from the nonce τ. Then the signature will be a zero-knowledge proof of the statement "I know the keys k, k_τ such that $\mathsf{Enc}_{k_\tau}(\mathsf{Enc}_k(x)) = y_\tau$". It's easy to see that this simple mechanism gets us most of the way towards key-blinding functionality. Anyone who knows τ and the previous public key y can derive the new public key y_τ simply by encrypting. Furthermore, the new y_τ is entirely disconnected from y under standard assumptions on the security of the encryption scheme. Anyone who knows τ and y will be unable to sign messages, even though this gives them k_τ as they do not have k and thus cannot construct the signature.

One issue with this system is that while y may be entirely changed by the blinding process, x does not change. If we keep the current system of having the public key system consist of both x and y, with x generated randomly by each user, then the unlinkability of the system is trivially broken, as the x component will not change. To fix this, we must rely

```
blPicnic.KeyGen()
1:  $k \leftarrow\!\!\$\ \{0,1\}^\lambda$
2:  $y \leftarrow \mathsf{Enc}_k(x)$
    // $x$ is a system parameter
3:  return $(pk, sk) = (y, k)$
```

```
blPicnic.BlindPk($y, \tau$)
1:  $k_\tau \leftarrow KDF(y \| \tau)$
2:  $y_\tau \leftarrow \mathsf{Enc}_{k_\tau}(y)$
3:  return $pk_\tau = y_\tau$
```

Figure 2.18: Key generation and blinding algorithms for blinded Picnic (blPicnic) signature scheme.

on the same technique that we did in the lattice case: x must be a fixed system parameter shared by all users.

In Figure 2.18 we describe key generation and signing in blPicnic. The general idea behind the blPicnic Sign and Verify algorithms does not substantially change from Picnic's sign and verification functions. The difference is that the circuit for security is twice as long to perform two encryption functions. As MPC-in-the-head protocols like ZKB++ can handle arbitrary circuits and inputs, this does not change what is possible, just the efficiency.

2.8.1 blPicnic Proof Outline

Here we outline the proof of unlinkability for blPicnic. For the independent blinding property, we need to show that

$$\mathsf{RandBlind}(pk_0; r_1), \ldots, \mathsf{RandBlind}(pk_0; r_n)$$

is indistinguishable from

$$\mathsf{RandBlind}(pk_1; r_1), \ldots, \mathsf{RandBlind}(pk_n; r_n),$$

where for a public key $y = \mathsf{Enc}_k(x)$, we define $\mathsf{RandBlind}(y, r)$ as just $\mathsf{Enc}_r(y)$. From this it is clear why the proof of unlinkability applies. To distinguish the two distributions is to distinguish many encryptions of the same plaintext versus many encryptions of different plaintexts. This reduces to the indistinguishability of the encryption scheme used. We can proceed by a simple game-hopping argument, where in game i we swap $\mathsf{RandBlind}(pk_0; r_i)$ for $\mathsf{RandBlind}(pk_i; r_i)$. As long as the ciphertexts are indistinguishable, the hop is justified, and we get that the advantage in breaking the independent blinding property is less than n times the advantage in breaking the indistinguishability of the ciphertext.

The proof of the signing with oracle reprogramming is similar to all Fiat–Shamir schemes. While we do not include all of the details (because we do not explicitly describe the signing protocol), we note that it is essentially the same as the proof for blLegRoast. Proceed through the protocol, but choose which parties will ‘cheat’ in advance. Then, set the outputs of those parties accordingly so that all checks pass. Reprogram the oracle so that those parties internal states stay secret. All checks will pass and the distribution of the signatures and reprogrammed points is identical.

The proof of the unforgeability of the scheme is similarly largely unchanged. The zero-knowledge protocol is meant to work for arbitrary circuits, so it is still the case that signatures do not leak any information about the secret keys.

Note however, that the statement that is being proven has changed. Before, signatures proved knowledge of just a k such that $\mathsf{Enc}_k(x) = y$, but now we are encrypting twice, that is, proving knowledge of two keys (k_τ, k) such that $\mathsf{Enc}_{k_\tau}(\mathsf{Enc}_k(x)) = y_{tau}$. This is a subtly different underlying problem, and its difficulty must be properly analyzed. Indeed, it is an *easier* problem to solve, as double encryption opens the possibility of meet-in-the-middle attacks.

If the length of the block is ℓ bits, then an adversary can create a list of $2^{\ell/2}$ keys k' and the associated value $\mathsf{Enc}_{k'}(x)$. By then sampling k'_τ values and seeing if $\mathsf{Dec}_{k'_\tau}(y_\tau)$ appears in the list, an adversary can find a secret key that allows them to construct forgeries in time roughly $O(2^{\ell/2})$. Thus, double encryption impacts the security of the underlying one-way function, and the block length ℓ may need to be proportionally increased as a result.

2.9 Implementation Discussion

We implemented the blDilithium-QROM, the blCSI-FiSh, and blLegRoast schemes; code for each is available at `http://github.com/tedeaton/pq-key-blinding`. The code for blCSI-FiSh and blLegRoast is forked from the CSI-FiSh and LegRoast code respectively [37, 35] and is written primarily in C. The code for blDilithium-QROM is written in Sage. Results can be seen in Table 2.1 at the beginning of the chapter. Our performance metrics indicate that the increase over the unblinded version of schemes is quite reasonable.

blDilithium-QROM. For blDilithium-QROM, key generation and verification are in fact *faster* since a fixed parameter $\mathbf{A}$ is used for all users and can be pre-generated, rather than being pseudorandomly generated each time. The signing procedure of blDilithium-QROM is three times slower than that of Dilithium-QROM. We caution that, since our blDilithium-QROM implementation is written in Sage, the implementation is non-optimized and results not be used an absolute measure of performance, but can still give insight when *compared* to a similar Sage implementation of non-blinded Dilithium-QROM.

blLegRoast. Blinded LegRoast's performance is compelling both in absolute terms (under 1 ms for key generation and blinding, under 20 ms for signing and verifying) and comparative terms (no worse than 1.5× slower than unblinded LegRoast).

blPicnic. We leave an implementation of blPicnic as future work. New advancements to the zero knowledge protocol that Picnic uses are still being made [19], so the performance of the scheme, and any blinded version, will change. We can summarize what we expect to see in a blPicnic implementation however. Public keys could keep the same size. The block size needs to be doubled to account for the meet-in-the-middle attack mentioned in Section 2.8.1, but the plaintext is a fixed value and does not need to be communicated. We do not have exact calculations for the signature size, but the circuit being used is twice as large (for two encryptions), so combined with the larger block size we might expect roughly four times as large. In practice it may not be quite this large however, as some of the values sent are independent of the length of the circuit.

blCSI-FiSh. Our blCSI-FiSh implementation achieves sizes and performance effectively matching that of CSI-FiSh-unMerkleized. The CSI-FiSh and blCSI-FiSh implementations use the CSIDH-512 parameter set. This parameter set aims to achieve NIST level 1 security (comparable to the security of AES-128 against a quantum adversary), though whether it achieves this level of security has been a matter of contention [123]. Unfortunately, increasing the parameters in CSI-FiSh is a matter of great difficulty. It is essential to CSI-FiSh that the structure of the class group be known. Calculating the order N of the group was a subexponential computation that took the CSIDH authors 52 core years. If the parameters are increased, then a new computation must happen, which will almost certainly be infeasible. Quantum computers could calculate the structure of the class group much more easily, so by the time CSI-FiSh is needed, there may also be the ability to use it by computing the class group number.[2]

2.10 Chapter Conclusion

We have considered the problem of building post-quantum key blinding schemes. We have shown that the unlinkability property can be reduced to two properties that are

[2]Crucially, this is because CSI-FiSh is an authentication protocol and thus using a quantum computer to decrypt past messages isn't a concern. In the next chapter we'll look at using CSI-FiSh for key establishment, where this poses more of an issue.

often relatively easy to establish: that blinding properly re-randomizes the public key (independent blinding) and that the distribution of signatures is only dependent on the public key (signing with oracle reprogramming). We have shown four different ways that post-quantum key blinding can be achieved: with supersingular isogenies via CSI-FiSh, lattices via Dilithium-QROM, with only symmetric primitives via Picnic, and by a number theoretic construction via LegRoast. We implemented blDilithium-QROM, blCSI-FiSh, and blLegRoast, and saw small performance impact compared to the unblinded versions.

Each of these four schemes is built out of the Fiat–Shamir paradigm. We did not consider any schemes built out of other ways to build signature schemes, such as hash-based signatures like SPHINCS+ [31], or the hash-and-sign paradigm like Rainbow [62] or Falcon [127].

It is difficult to envision a hash-based key blinding scheme. As public keys are the root of a Merkle tree, the only simple operation to blind a public key would be to hash it again. This could satisfy independent blinding, but not signing with oracle reprogramming: hash-based signatures work by providing paths up to the root, so the identity public key would be revealed on that path.

Hash-and-sign algorithms appear to have the opposite problem. A blinded version would almost certainly satisfy the signing with oracle programming property. If the trapdoored function is F, then by choosing a point x in the domain of F and programming the hash function so that $H(msg) = F(x)$, we obtain a signature; this is how hash-and-sign signature schemes often prove security. But it is not clear how to justify the independent blinding property. The most simple blinding mechanism would be to compose the trapdoor function F with another mapping G based on the blinding factor. This requires the range of F to match the domain of G, which makes it an interesting problem to be used with a hash-and-sign scheme. As well, to ensure the independent blinding property, we need that $F \circ G$ cannot be decomposed into the two mappings, which is a more novel security assumption. Because RSA is a trapdoor *permutation*, the structure of its mapping may allow for key-blinding, but it is not clear if any post-quantum primitive immediately does.

For these reasons, signature schemes that follow the Fiat–Shamir paradigm appear to admit key blinding much more readily. While homomorphic properties over the key space are certainly useful for key blinding (as in Dilithium and CSI-FiSh), they are not actually necessary, as the Picnic construction shows.

What this means for key-blinding in Tor. Recall that the motivation for this chapter was to provide a post-quantum alternative to key-blinding as it is used in Tor's onion service rendezvous protocol. For onion services, identity public keys are the URL for `.onion` addresses. This means that, unless the onion service lookup process changes, users directly interact with an onion service's public key, whether by clicking on it as a link or copying and pasting it into a browser window. This motivates keeping public keys as small as possible. For this purpose, blPicnic and blLegRoast are the most attractive of the schemes considered. In the context of Tor, the process for connecting to an onion service is quite lengthy (several seconds, usually). This is due to the numerous intermediate connections that must be resolved to build a connection. As a result, there may be less sensitivity to increased computation time.

Chapter 3

Post-Quantum Updatable Public-Key Encryption

3.1 Introduction

Secure communication protocols are quickly evolving [17, 111, 112], driven by the need to meet simultaneous usability and security requirements, such as asynchronous communication, forward secrecy, and post-compromise security for conversations that may last months, if not years. *Key-updatable public-key encryption* (UPKE) schemes have been proposed as a solution to improve weak forward secrecy properties of continuous key agreement schemes that underpin existing secure messaging protocols such as the Message Layer Security (MLS) protocols [9, 15, 95, 101, 125, 126]. In addition to standard public-key encryption functionality, UPKE schemes allow encryption and decryption keys to be asynchronously *updated* with fresh entropy. This can have the potential to *heal* the protocol by restoring security even after exposure of secret values. Unfortunately, the security of all UPKE schemes proposed to date relies on the hardness of the discrete logarithm problem.

Chapter Contributions and Structure. In this chapter, we perform the first assessment of the viability of quantum-secure UPKE schemes. We focus on the functionality of *symmetric UPKE*[1], and assess the extent to which the isogeny-based cryptosystem CSIDH [49] can be used to instantiate it. We model symmetric UPKE after a construction by Alwen et al. [9] to improve the forward-secrecy and post-compromise security of

[1]Symmetric here refers to the requirements of how the update operation is performed, not the style of encryption.

TreeKEM [39], the group key-exchange primitive used by MLS, where both encryption and decryption keys are updated using the *same* secret update value. Further, we introduce the notion of IND-CPA-U security, a generalization of a security model by Alwen et al. [9] for UPKE constructions.

We then present a CSIDH-based symmetric UPKE construction which can be used today with the existing CSIDH-512 parameter set, or any CSIDH parameter set where the class group structure is fully known. Knowing the class group structure ensures unique group element representation and uniform sampling of secret key material. Taken together, these properties ensure that knowledge of a secret key prior to an update will not leak information about the key *after* an update operation, thereby fulfilling forward secrecy and post-compromise security. We prove that our CSIDH construction fulfils IND-CPA-U security.

Related Work The most closely related work to our own is the already mentioned work of Alwen et al. [9] as a mechanism to improve forward secrecy and post-compromise security of TreeKEM [39]. Our symmetric UPKE primitive is modelled after their construction, and our work is an effort to define a post-quantum UPKE variant suitable for similar use. Further, we prove security in a more robust model that models the adversary's capability to both adaptively choose update values for the victim as well as corrupt their local state. The work by Alwen et al. was in turn based upon work by Jost et al. [101], which relies on a notion of 'asymmetric' UPKE.

Both secure messaging protocols and post-quantum protocols are still in active development. Efforts to combine the two into post-quantum secure messaging are so far rare in the literature, although we refer to [45] as a recent example of exactly this. Their work constructs a version of Signal's X3DH protocol out of the (ring)-learning with errors problem.

More recently, work by Dodis et al. [63] has directly constructed two UPKE schemes, based on the learning with errors problem (making it another example of a post-quantum construction) and the decisional Diffie-Hellman problem. A key focus of their work was to prove the security of their schemes in the *standard model*, as opposed to the random oracle model.

Alternative (Unrelated) Notions of UPKE. There exists a separate notion of 'updatable encryption' in the literature [41, 100]. In these schemes, a *ciphertext* is updated using an update token such that the encrypted message becomes an encryption under a new public key without decrypting the message. These schemes should not be confused with *key-updatable* UPKE schemes.

3.2 Background

3.2.1 CSIDH and CSI-FiSh

We have previously discussed CSIDH and CSI-FiSh in this thesis (Section 2.6). To reiterate, CSIDH instantiates a free and transitive *group action*, with the group being the class group $\mathrm{Cl}(\mathcal{O})$, with $\mathcal{O}$ being the endomorphism ring $\mathrm{End}_{\mathbb{F}_p}(E)$, the ring of endomorphisms from an elliptic curve E to itself defined over $\mathbb{F}_p$, which is an order in the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$. CSIDH chooses p to be of the form $4 \cdot p_1 \cdot p_2 \dots p_\ell - 1$, with each p_i a small distinct prime. This means that supersingular curves over $\mathbb{F}_p$, having $p+1$ points, will have many $\mathbb{F}_p$-rational subgroups with order p_i. In CSIDH, computing with the group is done via a set of canonical generators $\{\mathfrak{g}_i\}_{i \in \ell}$, where $\mathfrak{g}_i$ is an ideal of the form $\langle p_i, \pi - 1 \rangle$, where π is the Frobenius endomorphism. To apply a generator $\mathfrak{g}_i$ is to find the subgroup of order p_i defined over $\mathbb{F}_p$ and use Vélu's formulae to calculate the action of the isogeny whose kernel is that subgroup.

In CSIDH, secret keys are represented by a vector of ℓ integers. For efficiency reasons, the integers are usually chosen to be within a bound B, for example, $B = 5$ so that all entries are between -5 and 5. Then the secret key $[e_1, e_2, \dots, e_\ell]$ represents the group element

$$\mathfrak{g}_1^{e_1} \mathfrak{g}_2^{e_2} \cdots \mathfrak{g}_\ell^{e_\ell}.$$

Since the group is commutative, we have that if g is represented by $[e_1, \dots, e_\ell]$ and h is represented by $[f_1, \dots, f_\ell]$ then $g \cdot h$ can be represented by $[e_1 + f_1, \dots, e_\ell + f_\ell]$. Note as well that since the group operation works by applying the $\mathfrak{g}_i$ one at a time, the cost of applying a secret key is dominated by the ℓ_1-norm of the secret key.

It has always been known that this representation of group elements does not preserve distinctness. The $\mathfrak{g}_i$ generators are 'overkill' in that they are sufficient but likely not necessary to generate the entire group. But the exact structure of the group is not easily calculated from the parameters of the system. It is a subexponential calculation to understand the precise structure of the group. In CSI-FiSh [38], the authors performed this calculation to find this structure for the CSIDH-512 parameter set. They found the value N such that $|\mathrm{Cl}(\mathcal{O})| = N$. It is the product of 5 primes, the smallest being 3, and is approximately 2^{257}.

As the group is abelian and its order is a product of distinct primes, it is cyclic and the group is isomorphic to $\mathbb{Z}_N$. This means that it has a single generator, and in fact $\mathfrak{g}_1$ is such a generator. This representation buys us two key aspects: first, we now *do* have a unique or 'canonical' representation of group elements, and second we can sample uniformly over the group. These properties are very nice for cryptographic purposes where secret material needs to be masked and then released, such as in zero-knowledge proofs.

The problem with using the $\mathbb{Z}_N$ representation of the group is that it is inefficient to use directly. This is another major contribution of CSI-FiSh. The authors establish a way to efficiently switch between the representation of a secret key as an element in $\mathbb{Z}_N$ and one in $\mathbb{Z}^\ell$. Their observation is that finding an efficient representation is essentially solving a lattice problem. The vectors $v \in \mathbb{Z}^\ell$ that correspond to 0 in $\mathbb{Z}_N$ form a lattice. Finding a short representation in $\mathbb{Z}^\ell$ means finding *some* representation (which is easily done), and then finding the closest vector in the lattice, so that the difference between the two has small norm. Thus, the authors use a closest vector problem (CVP) solver to try and find a suitable representation.

Thus the overall group action is instantiated as follows:

1. Convert the representation in $\mathbb{Z}_N$ to one in $\mathbb{Z}^\ell$.
2. Use a CVP algorithm to find an equivalent representation $[e_1, \ldots, e_\ell]$ with small ℓ_1 norm.
3. For $i \in \{1, \ldots, \ell\}$, apply the $\mathfrak{g}_i$ ideal e_i times by finding the subgroup of order p_i over $\mathbb{F}_p$ and iteratively applying Vélu's formulae.

3.2.2 TreeKEM and UPKE

TreeKEM is a recent protocol to instantiate *continuous group key agreement* (CGKA) in a way that communication scales logarithmically, rather than linearly or quadratically, with the number of group participants. The main data structure in TreeKEM is a binary tree, where participants in the group are situated as the leaf nodes and the shared group secret lies at the root. Every node other than the root has a public and secret key pair associated with it. Each participant knows the secret keys on the path from their leaf node to the root, and the public keys on the *co-path* (the siblings to the nodes on the direct path). This is where TreeKEM gets its logarithmic efficiency from—encrypting a message under the public key of a node ν allows for any participant in ν's subtree to decrypt the message.

Two of the desired security properties of a CGKA are forward secrecy (FS) and post-compromise security (PCS). Informally, forward secrecy ensures that *future* compromises do not allow for messages sent *now* to be decrypted, and post-compromise security similarly ensures that *past* compromises do not allow for current messages to be decrypted (so long as an uncompromised 'refresh' of key material has occurred). TreeKEM achieves both of these properties through an 'update' process that allows any user to replace the group secret at the root, as well as all of the keys on their direct path to the root. The update process works as follows:

1. The user samples a seed s_0, from which they generate (via a pseudo-random generator) another seed s_1 and a secret key sk_0. They calculate the corresponding pk_0 and set the keypair for their node to be (sk_0, pk_0).

2. From s_1 they generate the seed s_2 and secret key sk_1. The user can replace the key pair of their parent node with the new (sk_1, pk_1). They then also need to send s_1 to their sibling node (the first node on the co-path) so that the sibling node can generate sk_1, as well as pk_0, the user's new public key.

3. From here, the user can continue up the direct path to the root, replacing key pairs and sending seeds and public keys to users who fall under the co-path so that they, too, can replace key pairs.

4. Eventually the user will replace the root with the seed s_d, where d is the length of the path to the root. The root does not need a keypair associated with it.

Note that TreeKEM also includes methodologies for adding and removing users, but as these discussions do not relate to UPKE, we do not discuss them here.

It is fairly easy to characterize TreeKEM's post-compromise security properties. If an adversary obtains the secret state of a user, they obtain that user's secret key, and all the secret keys up to the root, and the group shared secret. But as soon as that user performs an update operation, all of those keys are replaced with entirely fresh keys. As long as the user is not compromised during this update process, the adversary loses their advantage.

Analyzing the forward secrecy guarantees are somewhat more subtle. In an analysis by Alwen et al. [9], the authors considered the notion of 'epochs' as the number of times any user has performed an update. Naturally, for each epoch, the root group secret has changed. For the scheme to be optimally forward secure, it should be the case that as soon as any user has performed an update (thus replacing the root secret), a compromise of any user does not allow for the decryption of messages under the previous epoch.

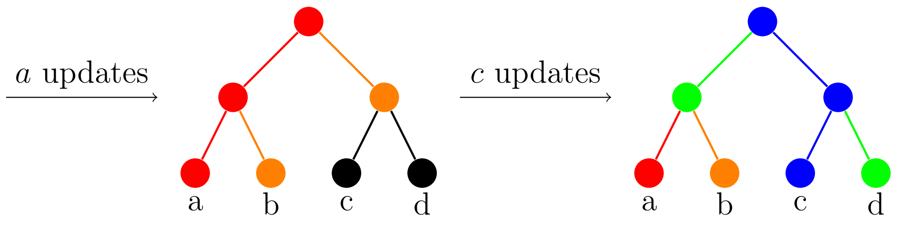


Figure 3.1: A minimal example to demonstrate how previous version of TreeKEM were not optimally forward secure.

Unfortunately, this is not the case. If the adversary captures all (encrypted) network traffic, then by compromising certain critical private keys the adversary can recover root nodes from previous epochs.

We consider an example in Figure 3.1. We begin with a tree with four parties, a through d. User a then performs an update action, replacing keys on their direct path to the root (in red) and encrypting seeds capable of generating parts of their direct path to the co-path (orange). Next, c performs an update, doing the same. We denote c's direct path in blue and their co-path in green. Then note that if party b's secret state is compromised in the second session, then the group secret in the first epoch can be recovered. This is simply because their own (leaf) keypair has not changed between the first and the second epoch, and as part of generating the first epoch, party a sent b a seed that generated the root secret under that keypair.

In Alwen et al.'s analysis [9], they show that in general, roughly half of the keys need to be updated before the forward secrecy property actually 'kicks in' for a given epoch. Their solution to this problem is that when an update occurs, both the direct path *and* the co-path are updated. This poses a problem however, as the updater is only meant to know the secret keys on the direct path.

The solution they propose is to use a construction they refer to as *updatable public key encryption* (UPKE). Their construction takes a CPA-secure KEM (with corresponding key generation, encapsulation, and decapsulation procedures) and adds an update procedure. The update procedure allows anyone to sample an 'update value' and apply it to the public key to generate an updated public key. With the same update value, anyone who has possession of the secret key can similarly update it to the new corresponding secret key. Critically, knowing (or even controlling) the update value still does not let you decrypt messages.

It is now clear how UPKE solves the problem of updating the co-path. When an update occurs, the user now *replaces* the direct path, and *updates* the co-path. They then know the secret keys on the direct path, but not the co-path. Since this corresponds to just a small amount of additional work for the co-path, the logarithmic scaling of TreeKEM is preserved, but forward secrecy is repaired.

3.2.3 Security

We now present a generalization of IND-CPA (Indistinguishability under Chosen Message Attack) security for UPKE schemes described by Alwen et al. [9], which we define as *indistinguishability under chosen plaintext attacks with updatability*, or IND-CPA-U. We present this notion of IND-CPA-U security in Figure 3.2. Our notion assumes a symmetric UPKE construction, but extends to the Asymmetric UPKE setting by simply allowing the adversary to learn public update values.

In Alwen et al.'s definition, the adversary is given the public key pk_0 and provides a sequence of updates $\mu_1, \ldots, \mu_\tau$. The public and private keys are updated accordingly and the adversary is issued an IND-CPA challenge under pk_τ. The public and secret key are updated again, this time with a secret update, and the adversary is given the resulting public *and* secret key. They then must respond to the IND-CPA challenge.

Their model illustrates the fundamental idea behind how security works for updatable encryption: the adversary may learn (or even control) either the update value or the secret key, but as long as they do not have both, the updated secret key remains secure. However they have the restriction that the adversary controls the updates prior to the IND-CPA challenge, and receives the secret key afterwards. We generalize this by allowing the adversary to adaptively choose whether they want to control the update or learn a secret key, with the restriction that the IND-CPA challenge can only be issued on a public key that has not been compromised in a straightforward way.

In Figure 3.2, we define the game. We first generate a keypair (pk_0, sk_0) and sending pk_0 to $\mathbf{Adv}$. We initialize $i \leftarrow 0$ (the most recent version of the keypair will be (pk_i, sk_i)). After this we let the adversary decide how the key will be updated. To this end, we provide our adversary with the following oracles:

- The $\mathsf{GiveUpdate}(\mu)$ oracle corresponds to an update that the adversary controls. They get to pick the update value μ, which is used to update the key pair. We keep track of the fact that the adversary provided the update so that if pk_{i-1} or pk_i becomes corrupted, the other does as well. While pk_i is provided to the adversary, note that they could also generate it themselves.

IND-CPA-U Game:

```
     // Derive starting keypair
1 :  $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(\lambda);\ i = 0$
2 :  $U \leftarrow \emptyset,\ C \leftarrow \emptyset$
     // Adversary queries oracles, eventually
     // returns a challenge index
3 :  $j \leftarrow \mathbf{Adv}^{\mathsf{GiveUpdate},\mathsf{FreshUpdate},\mathsf{Corrupt}}()$
     // Generate challenge ciphertext
4 :  $K_0 \leftarrow\!\$\ \mathsf{KeySp}$
5 :  $(c, K_1) \leftarrow \mathsf{Enc}(pk_j)$
6 :  $b \leftarrow\!\$\ \{0,1\}$
     // The adversary outputs a guess for $b$
7 :  $b' \leftarrow \mathbf{Adv}^{\mathsf{GiveUpdate},\mathsf{FreshUpdate},\mathsf{Corrupt}}(c, K_b)$
8 :  if  not $\mathsf{IsFresh}(j)$ return  $f \leftarrow\!\$\ \{0,1\}$
9 :  else return $b' = b$.
```

$\mathsf{GiveUpdate}(\mu)$

```
1 :  $i \leftarrow i + 1$
2 :  $pk_i \leftarrow \mathsf{UpdatePublic}(pk_{i-1}, \mu)$
3 :  $sk_i \leftarrow \mathsf{UpdatePrivate}(sk_{i-1}, \mu)$
     // Keep track of which updates Adv provided
4 :  $U \leftarrow U \cup \{i\}$
5 :  return $pk_i$
```

$\mathsf{FreshUpdate}()$

```
1 :  $i \leftarrow i + 1$
2 :  $\mu \leftarrow \mathsf{GenUpdate}()$
3 :  $pk_i \leftarrow \mathsf{UpdatePublic}(pk_{i-1}, \mu)$
4 :  $sk_i \leftarrow \mathsf{UpdatePrivate}(sk_{i-1}, \mu)$
5 :  return $pk_i$
```

$\mathsf{Corrupt}(j)$

```
1 :  $C \leftarrow C \cup \{j\}$
2 :  return $sk_j$
```

$\mathsf{IsFresh}(j)$

```
1 :  if $j \in C$ return false
2 :  $i \leftarrow j$
3 :  while $i \in U$
4 :    if $i - 1 \in C$ return false
5 :    $i \leftarrow i - 1$
6 :  $i \leftarrow j + 1$
7 :  while $i \in U$
8 :    if $i \in C$ return false
9 :    $i \leftarrow i + 1$
10 : return true
```

Figure 3.2: IND-CPA-U game definition.

- The $\mathsf{FreshUpdate}()$ oracle corresponds to updates happening that the adversary does not control or know the update value for. It generates a random $\mu \leftarrow\!\$\ \mathsf{GenUpdate}()$ and then calls $\mathsf{GiveUpdate}(\mu)$, providing the new pk_i to the adversary.
- The $\mathsf{Corrupt}(j)$ oracle provides sk_j for an index $j \leq i$, and notes that j was corrupted.

Eventually, the adversary requests a challenge on an index $j \leq i$. A random key K_0 is sampled and a genuine key and ciphertext is generated from $\mathsf{Enc}(pk_j)$. A bit $b \leftarrow\!\$\ \{0,1\}$ is sampled and (c, K_b) is provided back to the adversary. After making further queries to the update and corruption oracles, the adversary must issue a guess b'. They are said to win if $b = b'$ and the index j is *fresh*. The freshness requirement ensures that the adversary cannot trivially win.

An index j is considered fresh if:

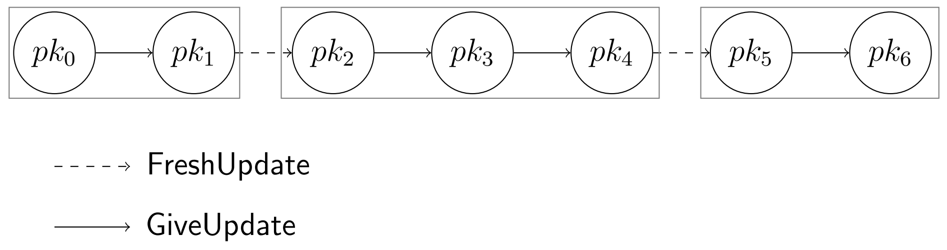


Figure 3.3: Visualizing how blocks of public keys become compromised depending on whether the adversary controls the update.

- The adversary has not called $\mathsf{Corrupt}(j)$, and
- There is not a sequence of adversary-provided updates (in either direction) that connects the index j to a corrupted index k.

In Figure 3.3 we visualize how the queries that the adversary has performed cause a given index to be considered fresh or not. When an adversary issues a $\mathsf{GiveUpdate}$ query, the new public key can be thought of as being in the same 'block' as the previous key, meaning that should one become compromised, the other will as well. $\mathsf{FreshUpdate}$ queries on the other hand result in a new block, disconnected from previous ones.

To emphasize, this definition means that if the adversary does not know the secret key associated with an index i, but then provides an update themselves and corrupts index $i + 1$, the index i is corrupted and sk_i can be easily calculated. One could hope for a system without this property, where corruption can only be propagated in the forward direction. This would be a notable improvement in the security of the system, but we leave it as an open question whether a scheme can be made to accomplish this property without weakening other parts of the security model.

Let **Adv wins** denote the event that the result of the IND-CPA-U game in Figure 3.2 outputs 1, that is the index is fresh and the adversary's guess for b was correct. We define the advantage of an adversary **Adv** against a UPKE scheme as $\left|\Pr[\mathbf{Adv\ wins}] - \frac{1}{2}\right|$.

Definition 13. A UPKE scheme is IND-CPA-U secure if for any polynomial-time adversary **Adv**, their advantage in the experiment in Figure 3.2 is negligible.

Unlike IND-CPA games for plain PKE schemes, the IND-CPA-U definition presented in Definition 13 captures the notion of forward secrecy and post-compromise security by allowing **Adv** to learn any secret key material and provide whatever update values that it wishes, with conditions preventing the adversary from trivially winning the IND-CPA game.

3.3 UPKE Construction

3.3.1 CSIDH UPKE, first attempt

A basic design for a symmetric UPKE scheme would then be for the update value to be a random group element, to update the public key by applying the group action, and to update the secret key by adding the group elements together.

Unfortunately, this simple design is not secure. If each entry for the update value is drawn uniformly from $-B$ to B, then the distribution of each entry of the new public key is centred at the old public key. This leaks a certain amount of information about the old secret key. For example, if only one update has occurred, and an entry is $2B$, then the adversary immediately knows that the corresponding entry before the update must have been B.

One fix may be to increase the bound B in an attempt to show that leaking the secret key between certain updates still doesn't reveal enough of the secret key to allow a break. Such an analysis must be done carefully, but reveals another fundamental problem. As more updates occur, the size of each entry in the vector is likely to grow. The efficiency of CSIDH is directly dependent on the ℓ_1-norm of this vector, and so allowing it to grow with updates will result in a slower and slower decryption process, eventually becoming unacceptable.

Note that this is almost exactly the same problem that a first attempt at a lattice-based scheme would run into. If one were to define a scheme based on the LWE problem, then updates could be generated by sampling an LWE secret. The secret key would then be updated by adding the update value to the old secret. But as described for CSIDH, this will cause the error term to grow over time, eventually causing the system to fail. Furthermore, because errors are not chosen uniformly, the distribution of a secret will always be dependent on the previous secret, meaning some information about previous keys is leaked in the event of a compromise.

One technique to circumvent this problem that has been to employ rejection sampling, as in the signature scheme SeaSign [60]. However, rejection sampling only works when we can reject the group elements that would leak information on the secret key. Since the party selecting the update value is not the owner of the public key, rejection sampling is not an option in our scenario. Instead, we will need the group elements to be represented in a way that has better properties.

As mentioned in Section 3.2.1, the signature scheme CSI-FiSh uses a different representation for group elements. Let N denote the order of the group. To compute the group action (i.e., apply the isogeny to an elliptic curve) one converts an element of $\mathbb{Z}_N$ (represented simply by an integer) to an ideal in $\mathbb{Z}^\ell$ and then applies the action as in CSIDH. Representing group elements as an integer in $\mathbb{Z}_N$ gives a unique representation. It is also still very easy to apply the group operation in this representation — it is just addition modulo N.

3.3.2 Our Construction

To prevent leakage from secret key updates described above, our construction requires a class group structure that is fully known, so that the secret key and update value can both be represented in $\mathbb{Z}_N$. The calculation of this value N as well as the methodology to convert the representation was a major contribution of the CSI-FiSh paper [38]. To update a public key, we apply the group action, and to update the secret key we add modulo N. Because we can sample uniformly over $\mathbb{Z}_N$, we have that the updated secret key leaks no information about the previous secret key, as desired.

We now describe the scheme in full, relying heavily on group action notation. Let N be the order of the class group $\mathrm{Cl}(\mathcal{O}) \cong \mathbb{Z}_N$. To apply the group action onto a supersingular elliptic curve E (denoted $g \star E$), we first need to convert the element to a representation in $\mathbb{Z}^\ell$ with a low ℓ_1 norm, and then apply the action as in the original CSIDH paper.

- $\mathsf{KeyGen}()$: Sample $g_{sk} \leftarrow_{\$} \mathbb{Z}_N$ and set $E_{pk} := g_{sk} \star E_0$. Output $(sk, pk) = (g_{sk}, E_{pk})$.
- $\mathsf{Enc}(pk)$: Sample $g_{enc} \leftarrow_{\$} \mathbb{Z}_N$ and compute $K \leftarrow \mathsf{KDF}(g_{enc} \star E_{pk})$, $E_{ct} \leftarrow g_{enc} \star E_0$. Return (E_{ct}, K).
- $\mathsf{Dec}(sk, E_{ct})$: Calculate and return $K' \leftarrow \mathsf{KDF}(g_{sk} \star E_{ct})$.
- $\mathsf{GenUpdate}()$: Sample $\mu \leftarrow_{\$} \mathbb{Z}_N$.
- $\mathsf{UpdatePrivate}(sk, \mu)$: Output $sk' \leftarrow sk + \mu \pmod N$.
- $\mathsf{UpdatePublic}(pk, \mu)$: Output $pk' \leftarrow \mu \star pk$.

Theorem 5. *Let* **Adv** *be an adversary capable of winning the IND-CPA-U game with advantage ϵ that makes q_{gen} queries to the* FreshUpdate *oracle. Then there exists an adversary capable of winning an IND-CPA game in time approximately equal to the running time of* **Adv** *with advantage $\epsilon/(q_{gen}+1)$.*

We demonstrate that our construction attains IND-CPA-U security, by showing a reduction from an adversary capable of winning the IND-CPA-U game to one that can win a plain IND-CPA game. By a plain IND-CPA game, we mean a game in which no calls to the GenUpdate, GiveUpdate, or Corrupt oracles are made. This corresponds to the IND-CPA security of CSIDH, except with secret key and encryption values drawn directly from $\mathbb{Z}_N$ and then converted to a vector to apply the group action. We present our complete proof in Section 3.3.3.

3.3.3 Proof of CSIDH-Based UPKE

We now present the proof of Theorem 5, the IND-CPA-U security of the CSI-FiSh-based construction.

Proof. As we are showing a reduction to a plain IND-CPA game, we will start by being given a public key pk^*. To begin, select a uniformly random index $i \leftarrow_{\$} \{0, \ldots, q_{gen}\}$. The idea of the proof is to set the public key after the ith FreshUpdate query to be pk^*, and hope that the adversary requests the IND-CPA-U challenge to be issued on a public key that occurs before the next FreshUpdate. If we are correct, then the adversary's ability to distinguish which message was encrypted under pk^* (or a related key) will allow us to win the IND-CPA game.

At the start of the game, if $i = 0$ then we set $pk_0 \rightarrow pk^*$. Otherwise, we sample a new uniform pk_0 from KeyGen. From here we proceed as normal. If the adversary makes a corruption query, then we provide them with the corresponding private key. When a GiveUpdate(μ) query is made, we update the secret and public key and make note of the μ value.

When the ith query to FreshUpdate is made, we set the resulting public key to pk^*. We carry on, and when the next FreshUpdate query is made we sample a fresh public key from KeyGen. We refer to the block of public keys (with their associated indices) between the ith and the $i+1$st FreshUpdate query as the target block. If the adversary ever makes a Corrupt query on any of the keys in the target block, or requests a challenge for an index *outside* the block, we abort. Because updates are sampled uniformly over $\mathbb{Z}_N$, the resulting public key is uniformly random over the public key space (this follows from the fact that the group action is regular). So after a FreshUpdate has occurred, the adversary has no information on the distribution of the secret key, and we can thus replace the public key with the challenge public key pk^*. The adversary has no advantage in distinguishing that we have done this. So the distribution of which block of public keys the adversary chooses to corrupt, and which one they may choose to issue a challenge for, is unchanged. There are $q_{gen} + 1$ blocks of public keys, so for the adversary to have any advantage over $\frac{1}{2}$, they must leave at least one uncorrupted and make their challenge query within that block. This means the chance that we abort is at most $q_{gen}/(1 + q_{gen})$.

Eventually, the adversary requests the IND-CPA-U challenge on a public key with index j, which by assumption is within the target block.

We then query for an IND-CPA challenge of our own, and receive back (C, K), with $C = g \star E_0$ for a random g, and K either a random key or $KDF(g_{sk^*} \star C)$, depending on the unknown bit b. Let $\mu_1, \mu_2, ..., \mu_k$ be k queries to GiveUpdate after the ith FreshUpdate query. We provide the adversary with $(-\mu_1 - \mu_2 - \cdots - \mu_k) \star C$ and K.

Note that if $b = 1$, then $K = \mathsf{KDF}(g \star pk^*) = \mathsf{KDF}((-\mu_1 - \cdots - \mu_k) \star g \star (\mu_1 + \cdots + \mu_k) \star pk^*)$, which means that the K we provide to the adversary has the right format. So, when the adversary submits a guess for b, we can guess the same value, and if the adversary is correct, so are we.

Our advantage in winning the IND-CPA game is thus the adversary's advantage in winning the IND-CPA-U game times the probability we do not abort, which is $\epsilon/(1 + q_{gen})$, as desired. $\square$

We note that the techniques in this proof can also be applied to the classical construction of Alwen et al. [9]. While they couple together the public key update and encryption functions, the same general strategy can be used to show that the stronger IND-CPA-U notion can be satisfied by their construction.

3.3.4 Implementation

Because uniform sampling and unique representation requires the structure of the class group to be known, our CSIDH-based scheme can only be instantiated with parameter sets for which that structure is known. At the present time, this requirement limits us to the CSIDH-512 parameter set, which claims 64 bits of post-quantum security. Peikert [123] has questioned this security claim, and more recent analysis [51] indicates that CSIDH-4096 is necessary for NIST level 1 security. Computing the structure of the class group is a sub-exponential computation, and so becomes feasible with the availability of a quantum computer to perform the computation. As such, the scheme may not be able to be instantiated until it is most needed.

Other than computing the class group, the main challenge in an implementation is in computing the group action. To compute the group action, the element of $\mathbb{Z}_N$ is converted to a vector in $\mathbb{Z}^\ell$, which represents the group element $\prod_{i=1}^{\ell} \mathfrak{g}_i^{e_i}$ for a vector $\vec{e}$ and set of generators $\{\mathfrak{g}_i\}_i$. This vector is then applied to the elliptic curve as is done in CSIDH.

Thus the additional complication over any other CSIDH implementation is in converting the element of $\mathbb{Z}_N$ to a vector of integers. This process is described in the CSI-FiSh paper, and the authors have provided code to do this (for the CSIDH-512 parameter set). The authors of CSI-FiSh found that the process of converting to a vector only makes a key negotiation 15% slower. Using their implementation of CSI-FiSh, we have a proof of concept script that illustrates the process of updating the secret and public keys. Our script is available at https://github.com/tedeaton/CSIDH-UPKE.

3.4 Chapter Conclusion

The post-quantum key-exchange protocol CSIDH instantiates a group action. In this chapter we have shown how a group action can be used to instantiate key-updatable public key encryption, a primitive used in continuous group key agreement protocol TreeKEM. We made use of several properties of this group action, most notably that it was free, transitive, that we could sample uniformly from the group and that we could give group elements a unique representation. CSIDH itself cannot provide all of these properties (it is missing unique group element representation), and so we needed to use the additional structure provided by calculating the size of the class group, as in the signature scheme CSI-FiSh.

In 'Cryptographic Group Actions and Applications' [6], the authors present a framework of group actions for cryptography, characterizing actions by the properties they provide. In their terminology, our construction for UPKE requires a 'Known-order Effective Group Action', or KEGA. Such group actions are among the strongest and hardest to instantiate because of the large amount of structure they require. CSI-FiSh in particular is difficult because it requires the size of the class group to be known. There is no efficient classical algorithm known for determining this size for a given parameter set. This is problematic, especially considering recent research that argues that the CSIDH-512 parameter provides insufficient protection against quantum computers compared to other post-quantum schemes [123].

One should note however that there are efficient *quantum* algorithms for determining the size of the class group. This places the scheme in a somewhat unique position: it can be instantiated securely, but only *after* large scale quantum computers already exist. The parameters require a quantum computer to generate, but once generated, the scheme can be efficiently run on any classical computer. This does not help with a 'pre-quantum' period of time, when quantum computers are imminent and quantum-secure cryptography should be used to prevent later retroactive mass decryption, but it may be part of a solution suite in a truly post-quantum era.

Chapter 4

Collision Finding in Non-Uniform Functions

4.1 Introduction

Hash functions are central and prominent in modern cryptography, and there have been many ingenious designs of cryptographic hash functions [120, 122, 32, 129]. One significant property of a cryptographic hash function H, backed with intensive tests in practice, is *collision resistance*. Namely, it should be computationally infeasible to find a *collision*, which is a pair of distinct input strings (x, x') with $H(x) = H(x')$. Because of this and other nice features, hash functions are used in numerous cryptographic constructions and applications, e.g., protecting passwords, constructing message authentication codes and digital signature schemes, as well as various crypto-currencies exemplified by Bitcoin [118].

Theoretical analysis of a hash function H often refers to *generic* security, where one ignores the internal design of H and views it as a black box. Moreover, the output of H is assumed to have been drawn *uniformly* at random from some codomain of size N. The complexity of finding a collision is then measured by the number of evaluations of H, i.e., queries to the black box. By the well-known *birthday* bound, $\Theta(\sqrt{N})$ queries are both sufficient and necessary to find a collision in H. These principles are extended and formalized as the *random oracle* model, in which a hash function is treated as a truly random function that is publicly available but only through oracle queries [24]. This heuristic has been widely adopted to construct more efficient cryptosystems and facilitate security reduction proofs which are otherwise challenging or unknown [25, 75].

But as with other areas of cryptography, quantum computers challenge long-standing results on the abilities of adversaries to compromise security. As discussed in Section 1.2, if H is treated as a black box, it is reasonable to allow a quantum adversary to query H in quantum superposition: $\sum_{x,y} \alpha_{x,y}|x, y\rangle \mapsto \sum_{x,y} \alpha_{x,y}|x, H(x) \oplus y\rangle$. In a 2015 paper, Zhandry [150] built on previous results to show that $\Theta(N^{1/3})$ quantum queries are both sufficient and necessary to find a collision in a uniformly random function. This establishes the generic collision resistance of uniformly random hash functions. But strict uniformity shouldn't be a necessary condition for security. Classically it is straightforward to show that any function with a large enough codomain and high enough entropy can still be collision resistant. Relaxing the condition of uniformity allows us to establish security for a wider range of functions, and understand how changes from uniform to non-uniform functions affect the abilities of quantum adversaries.

This motivates the question we study in this chapter: *what is the complexity of finding a collision in a* ***non-uniform*** *random function, under quantum attacks in particular*? Specifically we consider a distribution D_k on set Y which has min-entropy k, i.e., the most likely element occurs with probability 2^{-k}. We want to find a collision in a function $H : X \to Y$ where for each $x \in X$, $H(x)$ is drawn independently according to D_k. We call it a rand-min-k function hereafter. Note that if D_k is uniform over Y (hence $|Y| = 2^k$), this becomes the standard uniformly random function. Given H as a black-box, we are interested in the number of queries needed by a quantum algorithm to find a collision in H. As a result, this will establish the generic security of hash functions under a *relaxed condition* where the outputs of a hash function are drawn from a distribution of min-entropy k rather than a strictly uniform distribution. This condition might be a more realistic heuristic for a good hash function. Roughly speaking, a hash function designer will only need to make sure that there is no single value $y \in Y$ that has a large set of preimages (i.e., $f^{-1}(y) := \{x \in X : f(x) = y\}$ with $|f^{-1}(y)| \leq |X|/2^k$). In contrast, modelling a hash function as a uniformly random function would require certain *regularity* such that the preimage set of every codomain element has roughly the same size, which may be difficult to justify and test in practice. We also note that a concrete application of collision finding in rand-min-k functions appears in the famous Fujisaki–Okamoto transformation [75], whose quantum security has been studied in [137].

As noted, classically it is not difficult to derive a variation of the birthday bound, which gives $\Theta(2^{k/2})$ as the query complexity in typical cases. In the quantum setting, Targhi et al. [136] prove that $\Omega(2^{k/9})$ queries are necessary for any quantum algorithm to find a collision with constant probability. Compared to the tight bound $2^{k/3}$ in the uniform case, the bound is unlikely to be optimal and the gap seems significant. In addition, no quantum algorithms are described or analyzed formally. Overall, our understanding of finding a collision in non-uniform random functions is far from satisfying as far as quantum attacks are concerned.

Chapter Contribution and Structure In this chapter, we characterize the complexity of finding collisions in a rand-min-k function when it is given as an oracle to a quantum algorithm. We are able to prove matching upper and lower bounds in many cases. The results are summarized in Table 4.1. In this table, $\beta := \frac{1}{\Pr[x=y:x,y \leftarrow D]}$ is the collision variable, which equals 2^k for flat-distributions (i.e., uniform on a subset of size 2^k), and lies in $[2^k, 2^{2k}]$ for δ-min-k distributions (i.e., peak at one element, and uniform elsewhere), as well as for general min-k distributions. As well, M refers to the size of the domain and N refers to the size of the codomain. Bounds for all cases except when $M = o(\beta^{1/2})$ are original to this work. As well, these bounds are made with the assumption that the adversary has exact knowledge of the distribution D_k, except where otherwise stated.

D_k	M	N	**Upper bound**	**Lower bound**	**Upper $\overset{?}{=}$ Lower**
All	$M = o(\beta^{1/2})$	$N \geq 2^k$	∞ (Lemma 15)	∞ (Lemma 15)	✓
All	$M = \Omega(\beta^{1/2})$	$N \geq 2^k$	$\beta^{1/3}$ (Thm. 11)	$2^{k/3}$ (Cor. 3)	✗
flat-k	$M = \Omega(2^{k/2})$	$N \geq 2^k$	$2^{k/3}$ (Thm. 11)	$2^{k/3}$ (Cor. 3)	✓
δ-min-k	$M = \Omega(N^{1/2})$	$2^k \leq N < 2^{3k/2}$	$N^{1/3}$ (Thm. 11)	$N^{1/3}$ (Cor. 4)	✓
δ-min-k	$M = \Omega(2^k)$	$2^{3k/2} \leq N < 2^{2k}$	$2^{k/2}$ (Thm. 12)	$2^{k/2}$ (Cor. 4)	✓
δ-min-k	$M = \Omega(2^k)$	$N \geq 2^{2k}$	$2^{k/2}$ (Thm. 12)	$2^{k/2}$ (Cor. 4 and 5)	✓

Table 4.1: Summary of quantum collision finding in rand-min-k functions.

A simple special case is the flat distribution, which is uniform on a subset of size 2^k. In this case, not surprisingly, the same bound $2^{k/3}$ for the uniform random function holds. Another special case, which represents the hardest instances, concerns the δ-min-k distributions, where there is a mode element with probability mass 2^{-k} and the remaining probability mass is distributed uniformly throughout the rest of the codomain. Here we show that $2^{k/2}$ queries are both sufficient and necessary. For general min-k distributions, the complexity is characterized by the *collision variable* $\beta(D)$ for a distribution D, which is the reciprocal of the probability that two independent samples from D collide. We prove a generic upper bound $\beta^{1/3}$, and a lower bound $2^{k/3}$. For comparison, classically one can show that the (generalized) birthday bound $\Theta(\beta^{1/2})$, which equals $\Theta(N^{1/2})$ for uniform distributions, precisely depicts the hardness of finding a collision.

For the generic lower bound $2^{k/3}$, in Section 4.3 we follow the natural idea of reducing from collision finding in uniform random functions (Theorem 8). We show that finding a collision in a uniformly random function of codomain size 2^k reduces to that in flat distributions, and then to general min-k distributions. Therefore the $2^{k/3}$ lower bound follows. This approach is in contrast to that in [136], where they basically extract close-to-uniform bits from the output of a rand-min-k function f by composing f with a universal hash function h. Note that a collision in f is also a collision in $h \circ f$. In addition, $h \circ f$ can be shown to be quantum indistinguishable from a uniformly random function by a general theorem of Zhandry [149], which relates sample-distinguishability to oracle-distinguishability. Therefore any adversary for rand-min-k can be turned into an adversary for $h \circ f$, contradicting the hardness for uniformly random functions. However, the discrepancy between $h \circ f$ and a uniformly random function gets accumulated and amplified in the sample-to-oracle lifting step, and this may explain the slackness in their lower bound $2^{k/9}$.

Instead, given an oracle f whose images are distributed according to a distribution D, our reductions employ a *redistribution function* to simulate an oracle f' whose images are distributed according to another distribution D' on Y'. A redistribution function r maps a pair $(x, f(x))$ to an element in Y', and r is sampled from a proper distribution such that $f'(x) := r(x, f(x))$ is distributed according to D', taking into account the random choice of f as well. We show algorithms for sampling appropriate redistribution functions, called *redistribution function samplers*, for the distributions we are concerned with. As a result, we can use an adversary for the collision-finding problem in D' to attack the collision-finding problem in D. To complete the reductions, we show that a collision found in the simulated oracle for f' will indeed be a valid collision in f with probability at least $1/2$.

Along the same lines, it is possible to demonstrate that collision-finding in δ-min-k distributions is the hardest case. In fact, we are able to establish rigorously a *strengthened* lower bound in this case (Theorem 9). Our proof proceeds by showing indistinguishability between a random δ-min-k function on a codomain of size N and a uniformly random function on the same codomain. Then the lower bound in the uniform case translates to a lower bound for the δ-min-k case. The exact bounds vary a bit for different relative sizes of N and k.

In Section 4.4 we consider upper bounds. Establishing upper bounds is relatively easy (Theorem 11). We adapt the quantum algorithm of [150] in the uniform case. Basically we partition the domain of a rand-min-k function f into subsets of proper size, so that when restricting f on each subset, there exists a collision with at least constant probability. Next, we can invoke the collision finding algorithm by Ambainis [10] on each restricted function, and with a few iterations, a collision will be found.

Moreover, we give alternative proofs showing the lower bound for δ-min-k distributions (Theorem 10) and upper bound for all min-k distributions (Theorem 12). They are helpful to provide more insight and explain the bounds intuitively. Specifically, we reduce an average-case search problem, of which the hardness has been studied [92], to finding a collision in a δ-min-k random function. On the other hand, when the mode element of a min-k distribution is known, we show that applying Grover's quantum search algorithm almost directly will find a collision within $O(2^{k/2})$ queries. This actually improves the algorithms above in some parameter settings.

Finally, in Section 4.5 we consider what happens when we apply our redistribution algorithms to (second-)preimage resistance. Doing this allows us to also show lower bounds for the number of queries needed to find a (second-)preimage of a general min-k function. This shows the flexibility and strength of the redistribution technique in proving results about non-uniform functions.

Discussion Collision finding is an important problem in quantum computing, and a considerable amount of work in this context exists. Brassard et al. [44] give a quantum algorithm that finds a collision in any two-to-one function $f : [M] \to [N]$ with $O(N^{1/3})$ quantum queries. Ambainis [10] gives an algorithm based on quantum random walks that finds a collision using $O(M^{2/3})$ queries whenever there is at least one collision in the function. Aaronson and Shi [1] and Ambainis [11] give an $\Omega(N^{1/3})$ lower bound for a two-to-one function f with the same domain and co-domain of size N. Yuen [146] proves an $\Omega(N^{1/5}/\text{poly}(\log N))$ lower bound for finding a collision in a uniformly random function with a codomain at least as large as the domain. This is later improved by Zhandry [150] to $\Theta(N^{1/3})$ for general domain and codomain as we mentioned earlier.

We stress that, typically in quantum computing literature, the lower bounds are proven for the worst-case scenario and with constant success probability. This in particular does not rule out adversaries that succeed with an inverse polynomial probability which is usually considered a break of a scheme in cryptography. Hence a more appropriate goal in cryptography would be showing the number of queries needed for achieving any (possibly low) success probability, or equivalently bounding above the success probability of any adversary with certain number of queries. Our results, as in [150, 136], are proven in the strong sense that is more appropriate in cryptographic settings.

Our work leaves many interesting possible directions for future work. One immediate unsatisfying feature of our reductions is that they may take a long time to implement. Can they be made time efficient? We have been mainly concerned with finding one collision; it is interesting to investigate the complexity of finding *multiple* collisions in a non-uniform random function. There are other important properties of hash functions such as preimage resistance and second-preimage resistance, which are both weaker than and implied by collision resistance. Hence, our lower bound results also demonstrate the hardness of finding a preimage and second preimage, though the bounds are not necessarily tight. In Section 4.5 we extend the techniques for collision resistance and prove tight bounds for preimage- and second-preimage resistance against quantum generic attacks. Finally, we note that a stronger notion for hash functions called *collapsing* has been proposed which is very useful in the quantum setting [143]. Roughly speaking, for a hash function to be collapsing means that if an adversary prepares a quantum register of a superposition of inputs to a hash function that all evaluate to the same output, then that adversary cannot tell when a measurement in the computational basis has been applied to that register. Can we prove that rand-min-k functions are collapsing? Note that a uniform random function is known to be collapsing, and more recently it has been shown that the sponge construction in SHA-3 is collapsing (in the quantum random oracle model) [57].

Independent work. In a concurrent and independent work by Ebrahimi and Unruh [72], they give twelve bounds for quantum collision finding of min-k random functions. They frame their results somewhat differently to ours, organizing their bounds according to how they are quantified (e.g., for all adversaries, there exists a distribution such that at least so many queries are needed). They also characterize, for each case, bounds in terms of both the min-entropy k *and* the collision variable β. Our bounds in Table 4.1 on the other hand, are arranged according to distribution. Translating between the two, one sees that our results match closely with theirs, and indeed, in most cases a similar proof approach was employed. An exception to this is in the lower bound in terms of the min-entropy, where we achieve a tighter $2^{k/3}$ compared to their $2^{k/5}$. This difference is due to a different approach—our redistribution function technique is able to achieve a tighter reduction compared to the 'levelling approach' they use.

4.2 Chapter Background

Here we introduce a few notations and definitions. We also discuss basic results concerning the collision probability and birthday bound in min-k distributions.

Let D be a discrete probability distribution on set Y defined by probability mass function $D(y) := \Pr_{z \leftarrow D}[z = y]$. The support of D is $\{y \in Y : D(y) > 0\}$. We denote $Y^X := \{f : X \to Y\}$ the set of functions for some domain X and codomain Y. The notation $f \leftarrow Y^X$ indicates that f is a function sampled uniformly from Y^X.

Definition 14 (Min-Entropy)**.** Let D be a distribution on set Y. D is said to have min-entropy k if $k = -\log_2(\max_{y \in Y}\{D(y)\})$. We refer to a distribution of min-entropy k as a min-k distribution or simply a k-distribution.

Definition 15 (Flat-k-Distribution)**.** We call a k-distribution D on set Y a flat-k-distribution, denoted $D_{k,\flat}$, if the support S of D has size exactly 2^k. It follows that $\forall y \in S$, $D(y) = 2^{-k}$.

Definition 16 (δ-k-Distribution)**.** We call a k-distribution D on set Y a δ-k-distribution if there is a unique mode element $m \in Y$ such that $\forall y \in Y$

$$D(y) = \begin{cases} 2^{-k} & \text{if } y = m\,; \\ \frac{1-2^{-k}}{|Y|-1} & \text{otherwise}\,. \end{cases}$$

We denote such a distribution $D_{k,\delta}$. It is implicit that $|Y| > 2^k$. The support of D is the entire set Y, and remaining probability mass $1 - 2^{-k}$ is distributed uniformly among all elements in Y other than the mode.

Definition 17 (Function of min-entropy k)**.** Let D be a min-k distribution on set Y. We define D^X to be the distribution on Y^X such that for every $x \in X$, its image is sampled independently according to D. $f \leftarrow D^X$ denotes sampling a function in this way, and we say that f is a function of min-entropy k.

Definition 18 (Collision problem)**.** Let $f \leftarrow D^X$ be a function of min-entropy k. A pair of elements $x_1 \in X$ and $x_2 \in X$ such that $x_1 \neq x_2$ and $f(x_1) = f(x_2)$ is called a *collision* in f. We refer to the problem of producing such a pair as the *collision finding problem* in D.

Definition 19 (Quantum oracle access)**.** A quantum oracle $\mathcal{O}$ for some function f implements a unitary transformation: $\sum \alpha_{x,y}|x, y\rangle \overset{\mathcal{O}}{\mapsto} \sum_{x,y} |x, y \oplus f(x)\rangle$. An algorithm $\mathcal{A}$ that makes (quantum superposition) queries to $\mathcal{O}$ is said to have quantum oracle access to f, and is denoted $\mathcal{A}^f$.

4.2.1 Collision probability and non-uniform birthday bound

Definition 20. The *collision probability* of a probability distribution D is defined to be the probability that two independent samples from D are equal. Namely

$$\mathrm{CP}(D) := \Pr_{y_1,y_2 \leftarrow D}[y_1 = y_2] = \sum_{y \in Y} D(y)^2\,.$$

We call $\beta(D) := \frac{1}{\mathrm{CP}(D)}$ the collision variable of D.

$\beta(D)$ will be an important variable determining the complexity of collision finding. In fact we can derive a birthday bound for collisions in an arbitrary distribution D in terms of $\beta(D)$, analogous to the case of uniform distributions, using a key lemma by Wiener [145].

Lemma 14. *([145, Theorem 3]) Let R_D be the random variable denoting the number of i.i.d. samples from a distribution D until a collision appears for the first time. Let $q \geq 1$ be an integer and $\gamma_q := \frac{q-1}{\sqrt{\beta(D)}}$*

$$\Pr(R_D > q) \leq e^{-\gamma_q}(1+\gamma_q)\,.$$

Corollary 2. *Let $y_1, \ldots, y_q$ be i.i.d. samples from D, and let $Col^q(D)$ be the event that $y_i = y_j$ for some $i, j \in [q]$. There is a constant $c > 2$ such that if $q \geq c\sqrt{\beta(D)}$, then $\Pr(Col^q(D)) \geq 2/3\,.$*

Proof. Let E be the event that $y_i = y_j$ for some $i, j \in [q]$. Then

$$\Pr[E] \geq 1 - \Pr[R_D > q] \geq 1 - e^{-\gamma_q}(1+\gamma_q) \geq 2/3\,,$$

when $q \geq c\sqrt{\beta(D)}$ because $\frac{1+\gamma_q}{e^{\gamma_q}} < 0.3$ whenever $\gamma_q = \frac{q-1}{\sqrt{\beta(D)}} > 2$. □

We can also derive an upper bound on $\Pr[Col^q(D)]$ in a straightforward manner.

Lemma 15. $\Pr[Col^q(D)] \leq \frac{q^2}{\beta(D)}$.

Proof. For any pair $i \in [q]$ and $j \in [q]$, let Col_{ij} be the event that $y_i = y_j$. Then $\Pr[Col_{ij}] = \mathrm{CP}(D)$. Therefore by the union bound, we have

$$\Pr[Col^q(D)] = \Pr[\cup_{i,j\in[q]} Col_{ij}] \leq \binom{q}{2} \cdot \mathrm{CP}(D) \leq \frac{q^2}{\beta(D)}\,.$$

□

As a result, when $q = o(\sqrt{\beta(D)})$, essentially no collision will occur. Namely q needs to be $\Omega(\sqrt{\beta(D)})$ to see a collision, which is also sufficient by Corollary 2. This is summarized below as a birthday bound for general distributions.

Theorem 6. $\Theta(\sqrt{\beta(D)})$ *samples according to D are sufficient and necessary to produce a collision with constant probability for any classical algorithms.*

Finally, we characterize $\beta(D)$ for min-k distributions.

Lemma 16. *Let D_k be a min-k distribution on Y with $|Y| = N \geq 2^k$ and $k \geq 1$.*

- *For a flat-k distribution $D_{k,\flat}$, $\beta(D_{k,\flat}) = 2^k$.*
- *For δ-min-k distribution $D_{k,\delta}$, $\beta(D_{k,\delta}) \approx \begin{cases} N & \text{if } N < 2^{2k}\,; \\ 2^{2k} & \text{if } N \geq 2^{2k}\,. \end{cases}$*
- *For a general min-k distribution D_k, $\beta(D_k) \in [2^k, 2^{2k}]$.*

Proof. For flat-k D_k, $D_k(y) = \frac{1}{2^k}$ for all $y \in Y' \subseteq Y$ with $|Y'| = 2^k$. Hence $\beta(D_k) = \frac{1}{\sum_{y \in Y'} 2^{-2k}} = 2^k$. For the $D_{k,\delta}$ distribution,

$$\beta(D_{k,\delta}) = \frac{1}{\mathrm{CP}(D_{k,\delta})} = \frac{1}{2^{-2k} + \frac{(1-2^{-k})^2}{N-1}} = \frac{2^{2k}(N-1)}{N - 2 \cdot 2^k + 2^{2k}} \approx \frac{2^{2k} \cdot N}{2^{2k} + N}\,.$$

Different ranges of N give the estimation for $\beta(D_{k,\delta})$. For general D_k, it is easy to see that $2^{-2k} \leq \mathrm{CP}(D_k) \leq 2^{-k}$ and hence $\beta(D_k) \in [2^k, 2^{2k}]$. □

4.3 Lower bounds: finding a collision is difficult

We prove our quantum query lower bounds for min-k collision finding by security reductions. Recall the hardness result for uniform distributions by Zhandry [150].

Lemma 17 ([150] Theorem 3.1)**.** *Let $f : [M] \to [N]$ be a uniformly random function. Then any algorithm making q quantum queries to f outputs a collision in f with probability at most $C(q+1)^3/N$ for some universal constant C.*

We show that collision finding in any min-k distribution is at least as difficult as collision finding in a uniform distribution on a set of size 2^k. We begin by demonstrating a reduction of collision finding in a uniform distribution to collision finding in a flat-k distribution. Then we show a reduction of collision finding in a flat-k distribution to collision finding in a general k-distribution. Therefore we prove the following results.

Theorem 7. *Let $f_\flat \leftarrow D_{k,\flat}^X$ be a random function whose outputs are chosen independently according to a flat-k-distribution $D_{k,\flat}$. Then any quantum algorithm making q queries to $f_\flat$ outputs a collision with probability at most $O((q+1)^3/2^k)$.*

Theorem 8. *Let $f_D \leftarrow D^X$ be a random function whose outputs are chosen independently according to a distribution D of min-entropy k. Then any quantum algorithm making q queries to f_D outputs a collision with probability at most $O((q+1)^3/2^k)$.*

Corollary 3. *Any quantum algorithm needs at least $\Omega(2^{k/3})$ queries to find a collision with constant probability in a random function $f_D \leftarrow D^X$ whose outputs are chosen according to a distribution D of min-entropy k.*

Each of the proofs describes an algorithm (i.e., a reduction) attempting to find a collision in a random function f to which it has oracle access. The reduction will run, as a subroutine, another algorithm which finds a collision in another random function g when given oracle access to g (these random functions are not necessarily sampled from the same distribution). To adapt the subroutine which finds collisions in g for the task of finding a collision in f, the reduction simulates an oracle for g by building an oracle converter from the oracle for f and a suitable redistribution function. In general the redistribution function must be random, sampled from a particular distribution so that the distribution of its images equals that of g. Given some distributions from which the images of f and g are sampled, only some special sampling procedures will produce a redistribution function suitable for building the oracle converter needed. We formalize the concept of a *redistribution function sampler* as a generally randomized algorithm that performs such a sampling procedure specific to the oracles the reduction has access to and needs to simulate.

Definition 21 ($D \to D'$ Redistribution Function Sampler)**.** Suppose $f : X \to Y$ is a random function whose images are distributed according to a distribution D. Let D' be a distribution on Y'. We call an algorithm S a $D \to D'$ *redistribution function sampler* if it returns a function $r : X \times Y \to Y'$ such that for all $x \in X$ and $y \in Y'$, $\Pr_{f,r}\left[r(x, f(x)) = y\right] = D'(y)$ and this probability is independent for each x.

We use the term *redistribution function* to refer to a function returned by a redistribution function sampler, explicitly stating the distributions when necessary. The redistribution function naturally induces an oracle converter.

Definition 22 (Oracle Converter)**.** Suppose $f \leftarrow D^X$ is a random function whose images are distributed according to a distribution D on Y. Let D' be a distribution on Y', and $r : X \times Y \to Y'$ be a $D \to D'$ redistribution function. An algorithm $\mathcal{C}$, having oracle access to f and r, is called an *oracle converter from f to g* if $\mathcal{C}$ computes a function $g : X \to Y'$ defined by $g(x) := r(x, f(x))$.

We may denote $g = \mathcal{C}_{f,r}$. We can immediately observe that g is distributed as if the images were sampled independently according to D', when f and r are sampled according to the above definition.

Lemma 18. *The oracle converter defined above computes a function g that is distributed identically to ${D'}^X$, i.e., its images are independently distributed according to D', if $f \leftarrow D^X$ is chosen randomly and r is generated by a $D \to D'$ redistribution function sampler.*

We will be concerned with finding collisions in f and g. In particular, we are interested in whether a collision of g constitutes a collision of f. We define the collision-conversion rate to capture this property of an oracle converter.

Definition 23 (Collision-conversion rate)**.** Let $\mathcal{C}$ be an oracle converter from f to g. We say that it has *collision-conversion rate p* if for any $x, x' \in X$, and $g' \in {Y'}^X$ we have that

$$\Pr_{f,r}\left[f(x) = f(x') \mid g(x) = g(x') \wedge g = g'\right] \geq p.$$

With these notions available, our reduction proofs basically sample a redistribution function (with the correct distributions), and then simulate a correct oracle g distributed according to ${D'}^X$ using an oracle converter accessing the given oracle $f \sim D^X$. Then we run a collision-finding adversary on D' with oracle g. Whenever it outputs a collision, we can conclude that a collision is also found in f with probability p by the collision-conversion rate, which will lead to the desired contradiction. For each of the reductions, we will describe a suitable redistribution function sampler and show that it has at least constant collision-conversion rate. To do so, we assume that the reductions have full information about D and D', as well as sufficient randomness. This is fine as far as query complexity is concerned, and it is an interesting open question to make them time-efficient. We also remark that, for the sake of clarity, the distribution of images of our redistribution function is defined to be *exactly* matching distribution D'. It suffices to approximate distribution D' up to some negligible statistical distance.

Now we provide a generic formal description for all of our reductions, leaving the redistribution function sampler as a modular component which we can describe individually for each collision finding problem (for now we assume that each reduction has access to an adequate redistribution function sampler in each case). We do this in part to formally demonstrate how our reductions are compatible with *quantum* adversaries, allowing them to submit queries in quantum superposition and receive the oracle responses in quantum superposition. We will show that the oracle converters can be implemented as quantum

```
    Let $f \leftarrow D^X$ be a random function whose images are sampled according to $D$ on a set $Y$.
    Let $D'$ be a distribution on a set $Y'$. Let $S$ be a $D \to D'$ redistribution function sampler.
    Let $\mathcal{A}$ be an adversary for collision-finding in $D'$.
1 : Run $S$ and store its output as $r$. Implement an oracle for $r$.
2 : Construct an oracle converter $\mathcal{C}$ using the oracles for $f$ and $r$. The responses of $\mathcal{C}$ are
    now distributed according to $D'$. Refer to the function implemented by $\mathcal{C}$ as $g$.
3 : Initialize $\mathcal{A}$. For each query made by $\mathcal{A}$, forward the query to $\mathcal{C}$ and return the
    response to $\mathcal{A}$.
4 : When $\mathcal{A}$ returns a collision $(x_1, x_2)$ in $g$, output $(x_1, x_2)$.
```

Figure 4.1: Generic Reduction via Oracle Converter

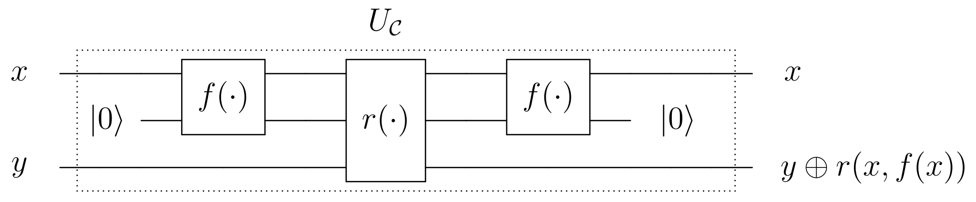


Figure 4.2: Quantum circuit that implements function $g = \mathcal{C}_{f,r}$ using two oracle calls to f.

oracles, so that the reduction can simulate the collision-finding problem for a quantum adversary who submit quantum queries. As usual, we consider a reduction solving collision-finding in D using an adversary for collision-finding in D'. The process is summarized in Figure 4.1.

We emphasize that the functions f and r are random functions sampled before the adversary begins the attack (the *attack* referring to the query-response phase in which interaction with the oracle occurs), as f is simply a model for what would be a fixed, publicly known hash function in a practical security setting, and r would be chosen by the adversary according to some procedure specific to the hash function (this is the role played by redistribution function sampler). Implementing the converter as a quantum-accessible oracle is straightforward as shown in Figure 4.2. Note that the function r can be turned into a unitary operator by standard technique $|x, \tilde{x}, y\rangle \overset{r}{\mapsto} |x, \tilde{x}, y \oplus r(x, \tilde{x})\rangle$. f is given as a quantum oracle, which we just need to query twice to answer each query to g.

Now that we have a generic construction for our reductions, we will show a simple reusable general result that will allow us to quickly construct reductions and extend query complexity lower bounds by simply demonstrating the existence of a satisfactory redistribution function sampler for use in each reduction. In this context we say that a reduction algorithm *succeeds* if the output pair indeed forms a collision in the given oracle function.

Lemma 19. *Suppose there exists an algorithm $\mathcal{A}$ which solves collision finding in a distribution D' with probability at least $P_{\mathcal{A}}$, using q queries to an oracle for a function g whose responses are distributed according to D'* [1]*. Suppose there exists a $D \to D'$ redistribution function sampler S such that the induced converter has collision-conversion rate at least p. Then the process in Figure 4.1 initialized with S and $\mathcal{A}$, denoted $\mathcal{R}_{S,A}$, solves collision finding in D with probability at least $p \cdot P_{\mathcal{A}}$ using $2q$ queries to an oracle for f whose images are distributed according to D.*

Proof. Lemma 19 follows from the suppositions stated, from Lemma 18, and from the definition of the collision-conversion rate (Definition 23). Let $g = f(x, r(x))$, and let $\mathcal{A}^g$ denote the adversary instantiated with g. We then need to consider, over the randomness in f, r, and $\mathcal{A}$, the probability that the output that $\mathcal{A}$ provides is a collision in f.

$$
\begin{aligned}
&\Pr_{f,r,\mathcal{A}}[f(x_1) = f(x_2) \mid (x_1, x_2) \leftarrow \mathcal{A}^g] \\
&= \Pr_{f,r,\mathcal{A}}[f(x_1) = f(x_2) \mid g(x_1) = g(x_2), (x_1, x_2) \leftarrow \mathcal{A}^g] \cdot \Pr_{f,r,\mathcal{A}}[g(x_1) = g(x_2) \mid (x_1, x_2) \leftarrow \mathcal{A}^g] \\
&= \sum_{g' \in Y^{X'}} \left(\Pr_{f,r,\mathcal{A}}[f(x_1) = f(x_2) | g(x_1) = g(x_2), g = g', (x_1, x_2) \leftarrow \mathcal{A}^g] \right. \\
&\qquad \left. \times \Pr_{f,r,\mathcal{A}}[g = g' \mid (x_1, x_2) \leftarrow \mathcal{A}^g] \right) P_{\mathcal{A}}
\end{aligned}
$$

Here we have used Lemma 18 to reduce $\Pr[g(x_1) = g(x_2) \mid (x_1, x_2) \leftarrow \mathcal{A}]$ to $P_{\mathcal{A}}$. For the first probability in the summand, note that the definition of collision conversion resistance means that for each g', for all x_1, x_2, this conditional probability is at least p. So the fact that (x_1, x_2) was generated by $\mathcal{A}^g$ is irrelevant, and we can reduce this to be simply p. Then we recombine the summand and get the result $p \cdot P_{\mathcal{A}}$. The observation that $\mathcal{R}_{S,A}$ uses twice the number of oracle queries as $\mathcal{A}$ proves the lemma.

□

[1]The probability $P_{\mathcal{A}}$ reflects the randomness of oracle's responses and of $\mathcal{A}$.

Therefore to prove Theorem 7 and 8, all that is left is to show suitable redistribution function samplers.

Lemma 20. *Let U_{2^k} be a uniform distribution on a set Y of size 2^k. Let $D_{k,\flat}$ be a flat-k distribution on a set Y_1, which has support $S_{k,\flat} \subseteq Y_1$, and D_k a general min-k distribution on a set Y_2. There exist $U_{2^k} \to D_{k,\flat}$ and $D_{k,\flat} \to D_k$ redistribution function samplers, and the induced oracle converters have collision-conversion rates at least 1/2.*

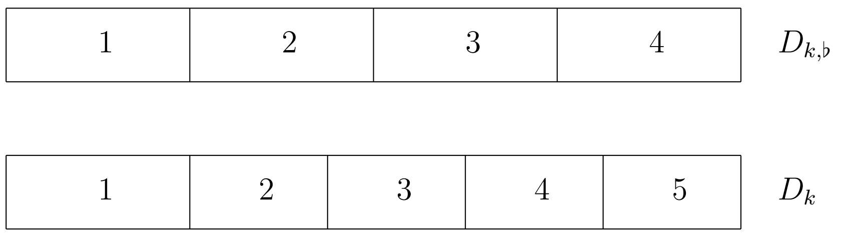
Proof. We describe the two samplers below.

$U_{2^k} \to D_{k,\flat}$ sampler. In this case the redistribution function sampler is nearly trivial because a simple relabeling of samples from the distribution U_{2^k} will suffice to simulate samples from the distribution $D_{k,\flat}$. Let f be a function $f : X \to Y$ whose images are distributed according to U_{2^k}, to which oracle access is available. Let $m : Y \to Y_1$ be any bijection onto $S_{k,\flat}$. Define S_1 as a one-step algorithm that returns a function $r_1(x, y) = m(y)$.

By the definition of r_1, $\Pr[r_1(x, f(x)) = y'] = \Pr[m(f(x)) = y']$ for all $x \in X$ and $y' \in S_{k,\flat}$. Since m implements an injective mapping from Y to $S_{k,\flat}$, $\Pr[m(f(x)) = y'] = \Pr[f(x) = m^{-1}(y')]$. Since, by the definition of f, $\Pr[f(x) = y] = U_{2^k}(y)$ for all $y \in Y$, $\Pr[f(x) = m^{-1}(y')] = U_{2^k}(m^{-1}(y')) = 2^{-k}$. Hence $\Pr[r_1(x, f(x)) = y'] = D_{k,\flat}(y')$ for all $x \in X$ and $y' \in S_{k,\flat}$, since $D_{k,\flat}(y') = 2^{-k}$ for all $y' \in S_{k,\flat}$. Anything not in the support of $D_{k,\flat}$ is also not in the range of m, and is returned with probability 0. It follows that S_1 is a $U_{2^k} \to D_{k,\flat}$ redistribution function sampler. We now show that the collision-conversion rate of the induced oracle converter is exactly 1. Let (x_1, x_2) be a collision in g, the function implemented by the oracle converter. Then $r_1(x_1, f(x_1)) = r_1(x_2, f(x_2))$, from which it follows that $m(f(x_1)) = m(f(x_2))$. Since m is an injective mapping,we can conclude that $f(x_1) = f(x_2)$, which shows that (x_1, x_2) is necessarily a collision in f.

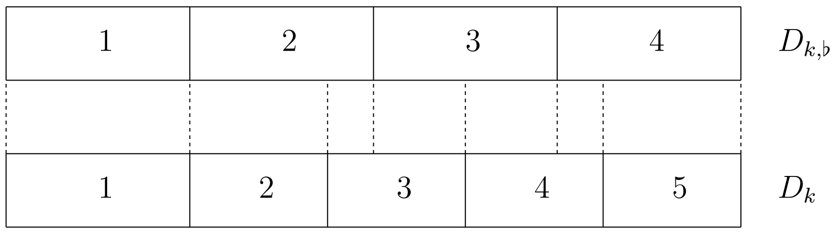
$D_{k,\flat} \to D_k$ sampler. We provide an overview of the $D_{k,\flat} \to D_k$ redistribution function sampler in the following few paragraphs. The complete redistribution function sampler is given in Section 4.3.1, along with a detailed explanation of the reasoning behind it. We reiterate that the redistribution function can be prepared before oracle access to the hash function under attack is obtained, allowing the query-response phase of the attack to be implemented as a quantum algorithm without concern for the quantum implementation of the redistribution function sampler.

The basic challenge that must be solved by the redistribution function sampler is to provide a mapping from the support of one distribution to the support of another distribution in such a way that the output is actually distributed according to the second distribution, which we call D_k, when the input is distributed according to the first, which we call $D_{k,\flat}$.[2] In order to maximize the probability that Algorithm 4.1 succeeds, the mapping must maximize the probability that two identical outputs correspond with two identical inputs, i.e., the collision-conversion rate. Our construction for this redistribution function sampler, which we call S_2 (and which returns a function which we call r_2), ensures that this probability is no less than one half by allowing at most two elements of the support of the $D_{k,\flat}$ be mapped to each element of the support of D_k. To provide intuition for how this is achieved, we recommend visualizing each distribution as a rectangle divided into 'bins' representing the elements of its support, with each bin's width proportional to the probability mass of the corresponding element under the distribution. We refer to this as the *rectangular representation* of the distribution. An example is shown below. We let $D_{k,\flat}$ be a flat distribution of min-entropy 2, and D_k be a (non-flat) distribution of min-entropy 2. We label each bin with a number indexing the elements of the support in each case.



For each of the elements of the support of $D_{k,\flat}$, we must decide what the probability mass corresponding to that element in $D_{k,\flat}$ should 'be sent to' by the redistribution function, in the sense that whatever that element is mapped to will occur with the same probability as that of sampling the element from $D_{k,\flat}$. A natural solution that would correctly produce the distribution D_k is to in some sense 'project' the distribution $D_{k,\flat}$ onto D_k, so that each 'location' in the rectangular representation of $D_{k,\flat}$ is mapped to a 'location' in the rectangular representation of D_k (by 'location' here we refer to horizontal position a rectangular representation, selecting some specific probability density). We illustrate this sort of projection by drawings lines between the two rectangular representations that show where the boundaries between the elements of each distribution's support fall in the other distribution, shown below.

[2]A redistribution function formally is also provided the query x that is associated with the sample from the first distribution, which is (in Algorithm 4.1) the response from an oracle whose output is distributed according to the first distribution. This is necessary in cases where the second distribution has a larger support than the first, since the image of the redistribution function cannot be larger than the domain. It can safely be ignored otherwise (as in the construction for r_1).



From the fact that the width of each bin is proportional to the probability mass associated with each element of each distribution, it follows that, if, for a given sample from $D_{k,\flat}$, we sample an element from the support of D_k according to the available probability mass *inside* the projected bin from $D_{k,\flat}$, the sampling result will be distributed exactly according to the distribution D_k. This is difficult to communicate verbally, but visually, one can imagine receiving a sample from $D_{k,\flat}$ as 'selecting' the bin associated with the sampled value in the rectangular representation of the distribution. Then, following the lines bordering that bin, we find that the probability mass associated with the sample from $D_{k,\flat}$ is mapped to probability mass corresponding to several elements of the support of distribution D_k. If we now sample from these elements according to their share of the probability mass corresponding to the sample from $D_{k,\flat}$, our samples will be distributed according to D_k. For example, with reference specifically to the graphic above, suppose that we receive element 2 as a sample from $D_{k,\flat}$. Following the lines down from the bin corresponding to element 2 in the rectangular representation of $D_{k,\flat}$, we see that elements 2 and 3 in the support of D_k both partially reside in the space corresponding to bin 2 in the rectangular representation of $D_{k,\flat}$. In particular, element 2 in the support of D_k consumes much more of the space than element 3. Hence we sample either 2 or 3, with a bias toward 2 exactly equal to how much more of the space element 2 consumes (recall that *space* in these rectangular representations corresponds to probability mass). Similarly, had we received element 3 as a sample from $D_{k,\flat}$, we would have sampled from elements 3 and 4 in the support of D_k with little or no bias, since these seem to roughly evenly split the space inside the boundaries of the bin corresponding to element 3 in the support of $D_{k,\flat}$.

In Section 4.3.1 we show a proof that the process in Figure 4.1 initialized with S_2 succeeds with probability at least one-half given that the adversary it runs as a subroutine succeeds—this proof derives and relies on a property of the redistribution function generated by the sampling procedure just described—that an apparent collision in its output corresponds with a collision in its input (which, recall, in Algorithm 4.1 is the output of an oracle whose images are distributed according to $D_{k,\flat}$) with probability at least one-half. In Section 4.3.2 we prove that the redistribution function sampler S_2 has a collision-conversion rate of at least one-half. The intuition behind this property is that a sample from D_k produced by the redistribution function could have been generated by, at most, 2 distinct samples from $D_{k,\flat}$, since each bin in the rectangular representation of D_k resides within the boundaries of, at most, 2 bins in the rectangular representation of $D_{k,\flat}$.

□

We have shown that S_1 and S_2, as just described, are $U_{2^k} \to D_{k,\flat}$ and $D_{k,\flat} \to D_k$ redistribution function samplers, respectively. Finally, Theorem 7 and 8 follow easily. Note that we write some of the constant factors in the probabilities with the common notation C, even though they will not all take the same numerical value, in recognition that they are not interesting for the study of asymptotic query complexity.

Proof of Theorem 7 & Theorem 8. By Lemma 20, there exists a $U_{2^k} \to D_{k,\flat}$ redistribution function sampler S_1 for which the induced collision-conversion rate is at least one-half. Therefore Lemma 19 implies that our reduction algorithm is an collision-finding adversary making $2q$ queries to a uniformly random function f with success probability at least $P_{\mathcal{A}}/2$. However, Lemma 17 tells us that any $2q$-query adversary can succeed with probability at most $C(2q+1)^3/2^k$. Therefore the success probability $P_{\mathcal{A}}$ of any q-query adversary $\mathcal{A}$ is $O((q+1)^3/2^k)$, which proves Theorem 7.

Theorem 8 is proved in the same fashion by invoking the $D_{k,\flat} \to D_k$ redistribution function sampler S_2 in Lemma 20 and with Theorem 7 taking the place of Lemma 17. □

4.3.1 Details relating to Lemma 20: $D_{k,\flat} \to D_k$ redistribution function sampler

In this section we describe the redistribution function sampler S_2 in complete detail. Annotated pseudocode for S_2 is found in Figures 4.3 and 4.4. Because the algorithm is quite long and dense, we use this section to also explain the intuition behind the algorithm, before proving that collision-conversion rate in Section 4.3.2.

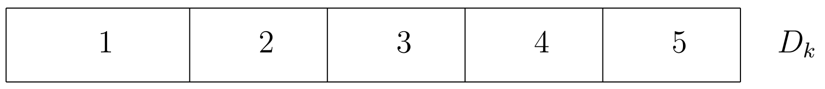
```
     Let $D_k$ be an arbitrary $k$-distribution on support $S_k$. Let $D_{k,\flat}$ be a
     flat-$k$-distribution on support $S_{k,\flat}$. Let $f_\flat : X \to S_{k,\flat}$ be an oracle for a function
     whose images are distributed according to $D_{k,\flat}$
1 :  Prepare to store a lookup table for a function $c : S_k \times S_{k,\flat} \to [0,1]$
2 :  Sort and label the elements $y_i$ of $S_k$ in order of decreasing probability mass under
     distribution $D_k$, so that $D_k(y_1) = 2^{-k}$ (by the definition of min-entropy, there must
     be one or more $y_i \in S_k$ with $D_k(y_i) = 2^{-k}$)
3 :  Arbitrarily label the elements $z_j$ of $S_{k,\flat}$ with index $j = 1, 2, \ldots, 2^k$.
4 :  for $i \in 1, 2, 3, \ldots, 2^k$ do
5 :    if $D_k(y_i) = 2^{-k}$ then
6 :      Set $c(y_i, z_i) = 1$.
7 :      Set $c(y_i, z_j) = 0$ for all $j \neq i$.
8 :    endif
9 :    if $D_k(y_i) < 2^{-k}$ then
10 :     Compute $\sum_{j=1}^{i-1} D_k(y_j)$ (here $j$ is a dummy-index for the sum and is unrelated to
       the labeling of the elements of $S_{k,\flat}$).
11 :     Save the result as the image of $i$ under a function $g : [|S_k|] \to [0,1]$.
12 :     if $2^{-k}$ divides $g(i)$ then
13 :       Set $c(y_i, z_{((g(i)/2^{-k})+1)}) = 2^k D_k(y_i)$ and $c(y_i, z_j) = 0$ for all $j \neq (g(i)/2^{-k}) + 1$.
14 :     else
15 :       Set $c(y_i, z_{\lceil g(i)/2^{-k} \rceil}) = 2^k \min(\lceil g(i)/2^{-k} \rceil - g(i), D_k(y_i))$
16 :       Set $c(y_i, z_{\lceil g(i)/2^{-k} \rceil + 1}) = 2^k (D_k(y_i) - \min(\lceil g(i)/2^{-k} \rceil - g(i), D_k(y_i)))$
17 :       Set $c(y_i, z_j) = 0$ for all remaining $z_j \in S_{k,\flat}$
18 :     endif
19 :   endif
20 : endfor
```

Figure 4.3: Redistribution Function Sampler S_2, Part 1.

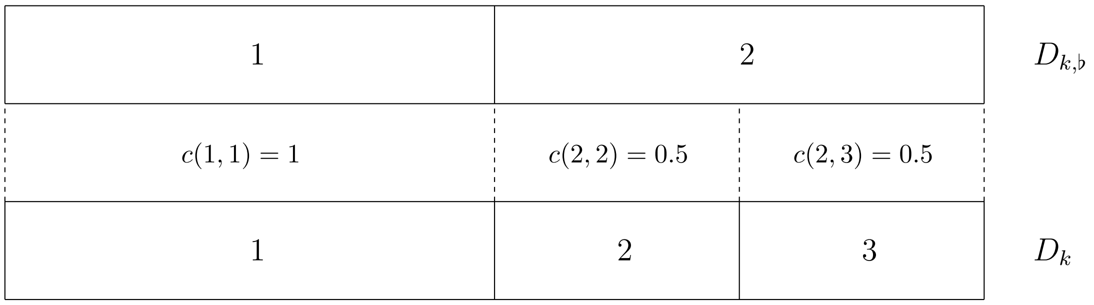
```
21 : for $z_j \in S_{k,\flat}$
22 :     Construct the set $W_j$ containing all $y_i \in S_k$ for which $c(y_i, z_j) \neq 0$.
23 :     Store the set $W_j$ as the image of $z_j$ in a function $b : S_{k,\flat} \to \mathcal{P}(S_k)$.
24 : endfor
25 : for each combination of one $x \in X$ and one $z \in S_{k,\flat}$ do
26 :     Fix some mapping from an index $t$ to each element in $b(z)$.
27 :     Compute $m_x(y_t) = c(y_t, z)$ for each $y_t \in b(z)$.
28 :     Sample $a \leftarrow [0, 1]$.
29 :     Iterate through $i = 1, \ldots, |b(z)|$ until $\sum_{t=1}^{i} m_x(y_t) \geq a$.
30 :     Define $r_2(x, z) = y_i$.
31 : endfor
32 : return $r_2$
```

Figure 4.4: Redistribution Function Sampler S_2, Part 2.

To facilitate intuitive understanding of why S_2 is a suitable redistribution function sampler for use in reducing collision finding in $D_{k,\flat}$ to collision finding in D_k, we visualize a distribution D_k as a rectangle divided into disjoint regions, each region representing one element of the support S_k. The total area of the rectangle is 1, representing the total probability mass in D_k. For simplicity, consider momentarily a distribution of min-entropy 2 on a set of size 5, whose elements we label with the first 5 positive integers. In the distribution represented below, the 1 element has 25% of the probability mass, while the others share the remaining 75%. Thus 1 is the mode element and its probability mass determines the min-entropy of the distribution.



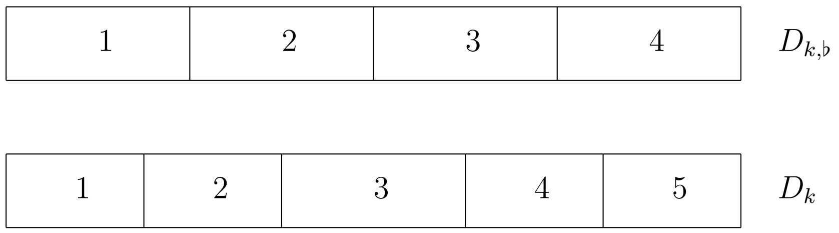
Denote the oracle simulated by the oracle converter (which we denote $\mathcal{C}$ as f_k. In order to be used in Algorithm 4.1, the redistribution function's output must be distributed according the distribution D_k when given a query $x \in X$ and a response from an oracle for $f_\flat$. Clearly, in order to satisfy this requirement, there must be additional randomness 'built in' to the way the redistribution function is sampled because the distribution D_k may, in general, have higher Shannon entropy than the distribution $D_{k,\flat}$, as $|S_k| \geq |S_{k,\flat}|$. We can accomplish this by treating the elements of $S_{k,\flat}$ as 'bins', each associated with one or more elements of S_k. In order to generate a sample from S_k, a sample from $S_{k,\flat}$ first selects a 'bin', and then one of the elements of S_k associated with that bin is chosen randomly according to a conditional probability distribution such that the marginal probability of sampling that element (across all of the bins) is equal to the probability associated with that element under distribution D_k. This process can be visualized intuitively by vertically aligning rectangular representations of the distributions $D_{k,\flat}$ and D_k. The conditional probability of sampling each element in S_k given a bin in $S_{k,\flat}$ can be illustrated by projecting the dividers between elements in the rectangle representing D_k into the space between the two rectangles. We show a trivial example below, using distributions of min-entropy 1.



The diagram above also illustrates the role played by the function $c : S_k \times S_{k,\flat} \to [0,1]$. This function specifies the conditional probability that r_2 returns a sample of a certain element of S_k given that the response of $f_\flat$ to a certain query is a certain element from $S_{k,\flat}$. Formally, $c(y,z) = \Pr[r_2(x, f_\flat(x)) = y | f_\flat(x) = z]$, where x is a query produced by the adversary in algorithm 4.1. Then the output of r_2 on input (x,z) is simply determined by sampling from the chosen bin according to the conditional probabilities that $c(y,z)$ specifies—this sampling step is where algorithm 4.3 injects the additional randomness that will be needed to convert an oracle for $f_\flat$ to an oracle for f_k. The values encoded into the function c therefore make up the non-trivial part of algorithm 4.3.

In the example above, the elements 2 and 3 have a total probability mass of 0.5 in D_k, so they can be 'binned' into element 2 in $D_{k,\flat}$. Supposing that element 2 is returned by the oracle $f_\flat$, a single uniformly random bit would determine whether $\mathcal{C}$ will return 2 or 3 as the response of f_k, since each are equally likely under D_k. If element 1 is returned by $f_\flat$, no additional randomness is needed, as $c(1,1) = 1$. This is always the case for the mode element, because its probability mass under D_k must exactly equal the probability mass of any of the elements of $D_{k,\flat}$, since D_k and $D_{k,\flat}$ have equal min-entropy.

This procedure will ensure that the output of r_2 is distributed according to D_k when the 2nd input is distributed according to $D_{k,\flat}$, since the marginal probability of sampling each element in this fashion exactly replicates the associated probability in D_k. In this simple case, any collision found in f_k will necessarily be a collision in $f_\flat$. However, this is not true in general. If the elements of S_k cannot be grouped into bins each with total probability mass under D_k equal to 2^{-k}, then some elements of S_k must have their probability mass split among multiple bins. An example of such a case is shown below.

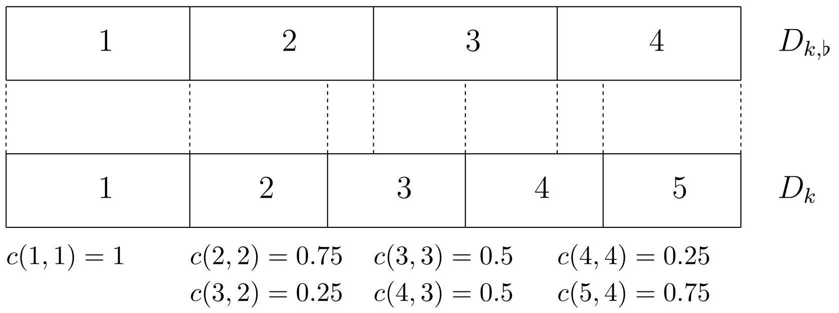


In cases like these, it is possible that a pair of identical outputs from r_2, constituting an apparent collision in f_k from the perspective of the adversary in algorithm 4.1, do not actually originate from identical responses from $f_\flat$, and therefore do not constitute an actual collision in $f_\flat$, the function which 4.1 attacks. Luckily, it is possible to construct the function c such that the probability that a collision in f_k corresponds to a collision in $f_\flat$ is bounded below by one half (giving $\mathcal{C}$ a collision-conversion rate of at least one half), so that Algorithm 4.1 instantiated with S_2 can preserve the query complexity of the adversary it leverages, from which Theorem 8 follows. Algorithm 4.3 contains a general method for constructing such a function, which we explain now.

The first step is to sort the elements of S_k in order of decreasing probability under distribution D_k. The utility of this is that it guarantees that any elements of S_k which can be trivially associated with an element in $S_{k,\flat}$, because they have a probability mass equal to 2^{-k}, are mapped to a single element in $S_{k,\flat}$ with probability 1.

Next, Algorithm 4.3 iterates over the elements y_i of S_k, setting the value of $c(y_i, z_j)$ for all elements z_j of $S_{k,\flat}$ for each. This may be visualized as moving across the rectangular representations of D_k and $D_{k,\flat}$ from left to right, determining the values of the function c along the way. If the probability mass of y_i under D_k is 2^{-k}, then all of its probability mass is associated with the element in $S_{k,\flat}$ with the same index, so $c(y_i, z_i) = 1$ (and of course zero for all other z_j). If the probability mass corresponding to y_i in D_k is less than 2^{-k}, then the probability mass from y_i will not 'occupy' an entire bin in $D_{k,\flat}$, so it must share a bin with other elements of S_k. But if the current bin is already partially occupied, it may be the case that the probability mass of y_i has to be split between the current bin and the next bin, where the 'current bin' and 'next bin' refer to the elements of $S_{k,\flat}$ which, at this point in execution of Algorithm 4.3, have the highest index out of the elements for which c has already been assigned and the lowest index out of the elements for which c has not already been assigned, respectively.

To check whether this is the case, Algorithm 4.3 computes the total probability mass in S_k that has already been assigned, which it saves as $g(i)$, and checks whether this value is a multiple of 2^{-k}. If it is, then the current bin must be completely occupied, and the probability mass corresponding to y_i will fit completely inside the next bin. The next bin will in this case be indexed by $(g(i)/2^{-k}) + 1$. If $g(i)$ is not a multiple of 2^{-k}, then the probability mass from element y_i may need to be split between the current bin and the next bin. In this case, the index of the current bin will be $\lceil g(i)/2^{-k} \rceil$, because for example if $g(i)/2^{-k} = 2.1$ then the first two bins are completely occupied, and one tenth of the third bin is completely occupied, making the index of the current bin 3. The index of the next bin will thus be the index of the current bin plus one. The value of $c(y_i, z_{\lceil g(i)/2^{-k} \rceil})$, representing the conditional probability of sampling y_i given the current bin has been sampled from $D_{k,\flat}$, is set to $2^k \min(\lceil g(i)/2^{-k} \rceil - g(i), P(y_i))$. The minimum function guarantees that if it is possible to fit all the probability mass from y_i into the current bin then this is done, and if not, whatever probability mass can fit into the current bin is assigned to the current bin. Naturally, whatever probability mass is not assigned to the current bin must be assigned to the next bin, to conserve marginal probability. The factors of 2^k come from the denominator of 2^{-k} in the conditional probability. It should be clear that the procedure just described would lead to a c function like the one illustrated below, for the example distribution D_k which we introduced earlier. In general, the c function that results from this procedure can be visualized by projecting the dividers between elements in *both* rectangles into the space between the two rectangles.



In the next step, for each element in $S_{k,\flat}$, Algorithm 4.3 saves the set of all elements in S_k which are associated with it via the c function. These sets are saved in a function b and will be used to 'invert' (using the term loosely) the c function for the purpose of simulating the responses of an oracle f_k whose responses are distributed according to D_k.

Now that the appropriate conditional probability distributions are calculated (in the c function), all Algorithm 4.3 has to do is sample from the conditional distribution specified by c for each possible combination of a query $x \in X$ and a response $z \in S_{k,\flat}$ of oracle $f_\flat$. Essentially, the steps before this point were defining the distribution over the set of all functions needed to sample a suitable redistribution function, and the remaining steps sample the function from this distribution, one image at a time. It is simple to verify that the last few steps in Algorithm 4.3 do exactly this via inverse transform sampling.

4.3.2 Proof that the oracle converter induced by S_2 has a collision-conversion rate of at least $\frac{1}{2}$

Let $D_{k,\flat}$ denote the flat-k-distribution on support $S_{k,\flat}$, and D_k denote a k-distribution on support S_k. Let $f_\flat$ be a function $f_\flat : X \to S_{k,\flat}$ whose images are distributed according to $D_{k,\flat}$, to which oracle access is available. Let $r_2 : X \times S_{k,\flat} \to S_k$ be defined as it is in algorithm 4.3. Let f_k be an arbitrary function in Y'^X, and g the oracle simulated by the oracle converter induced by f and r. To provide a bound on the collision conversion rate, take any x_1 and x_2. We then need to consider the probability that $f_\flat(x_1) = f_\flat(x_2)$, conditioned on the fact that $g = f_k$ and $g(x_1) = g(x_2)$.

Let y denote $f_k(x_1)$, this oracle's response to query x_1, and likewise let y' denote $f_k(x_2)$. Let z denote $f_\flat(x_1)$ and z' denote $f_\flat(x_2)$. Then the oracle converter's collision-conversion rate can be expressed as $\Pr_{f_\flat, r_2}[z = z' | g = f_k \wedge y = y']$. To help prevent confusion, we stress that the probability here is taken over the random choice of $f_\flat \leftarrow {D_{k,\flat}}^X$ and the randomness in 4.3 when preparing r_2.

We rewrite the constraint $g = f_k \wedge y = y'$ as $g = f_k^{y=y'}$, i.e., f_k is a function where $f_k(x_1) = f_k(x_2)$. Then by Bayes' theorem

$$\Pr[z = z' \mid g = f_k^{y=y'}] = \frac{\Pr_{f,r_2}[z = z']}{\Pr_{f,r_2}[g = f_k^{y=y'}]} \Pr_{f,r_2}[g = f_k^{y=y'} \mid z = z'] \tag{4.1}$$

Applying the law of total probability, we may write

$$\begin{aligned} \Pr[z = z'] &= \sum_{j=1}^{2^k} \Pr[z' = z_j | z = z_j] \cdot \Pr[z = z_j] \\ &= \sum_{j=1}^{2^k} \Pr[z' = z_j] \cdot \Pr[z = z_j] \quad (z \,\&\, z' \textit{ independent}) \\ &= \sum_{j=1}^{2^k} (2^{-k})^2 = 2^{-k} \quad \text{(definition of a flat-}k\text{-distribution)}\,. \end{aligned}$$

From Lemma 18, we also have that

$$\Pr_{f,r_2}[g = f_k^{y=y'}] = \prod_{x \in X} D_k(f_k^{y=y'}(x))$$

Then equation (4.1) can be rewritten as

$$\Pr[z = z' \mid g = f_k^{y=y'}] = \frac{2^{-k}}{\prod_{x \in X} D_k(f_k^{y=y'}(x))} \Pr[g = f_k^{y=y'} | z = z'] \tag{4.2}$$

Now we turn our attention to the conditional probability on the right. Applying the law of total conditional probability, we decompose the conditional probability into a sum over all possible values of the random variable z, so

$$\Pr[g = f_k^{y=y'} | z = z'] = \sum_{j=1}^{2^k} \Pr[g = f_k^{y=y'} | z = z' \wedge z = z_j] \cdot \Pr[z = z_j | z = z']$$

Applying Bayes' Theorem again, this time to the conditional expression on the far right, inside the summand, we get

$$\Pr[g = f_k^{y=y'}|z = z'] = \sum_{j=1}^{2^k} \Pr[g = f_k^{y=y'} \mid z = z' \wedge z = z_j] \cdot \frac{\Pr[z = z_j]}{\Pr[z = z']} \cdot \Pr[z = z'|z = z_j].$$

Using our prior result for $\Pr[z = z']$, and the fact that all samples according to $D_{k,\flat}$ have probability 2^{-k}, we can see that the ratio in the above equation must be exactly one. Hence we get

$$\begin{aligned}\Pr[g = f_k^{y=y'} \mid z = z'] &= \sum_{j=1}^{2^k} \Pr[g = f_k^{y=y'} \mid z = z' = z_j] \cdot \Pr[z' = z_j] \\ &= \sum_{j=1}^{2^k} \Pr[g = f_k^{y=y'}|z = z' = z_j] \cdot 2^{-k}\end{aligned}$$

Again thanks to Lemma 18 we can break apart $\Pr[g = f_k^{y=y'}|z = z' = z_j]$ into the product for each $x \in X$, as all are independent, except for x_1 and x_2. So we can rewrite this probability as

$$2^{-k} \left(\prod_{x \in X \setminus \{x_1, x_2\}} D_k(f_k^{y=y'}(x)) \right) \sum_{j=1}^{2^k} \Pr[y = y' = f_k^{y=y'}(x_1)|z = z' = z_j]$$

For convenience, set $f_k^{y=y'}(x_1) = y''$. Then we can rewrite the summation as

$$\sum_{j=1}^{2^k} \Pr[y' = y''|z' = z_j] \Pr[y = y''|z = z_j] \tag{4.3}$$

This step is only possible because y' and z are independent, so the presence of z' in the conditional involving y' can be ignored. The same goes for y and z' in the second conditional.

Now, recall that the function c is defined in algorithm 4.3 by $c(y, z) = \Pr[f_k(x) = y|f_\flat(x) = z]$. It follows, by the definitions of y, y', z, and z', that each of the factors in the summand are equal to $c(y'', z_j)$. Hence we may write equation 4.3 as

$$\sum_{j=1}^{2^k} c(y'', z_j)^2.$$

From the details of how the values of c are assigned in algorithm 4.3, the number of j values for which $c(y'', z_j)$ is non-zero is either one (if all of y'''s probability mass is associated with a single bin), or two (if the probability mass is split over two bins). If only one value of j corresponds to a non-zero $c(y'', z_j)$, then $c(y'', z_j)$ must be $2^k D_k(y'')$. But we are interested in the worst case, in order to establish a lower bound. If two values of j correspond to a non-zero $c(y'', z_j)$, then we know that the two values of c must sum to $2^k D_k(y'')$. In this case, the sum of the squares of these values will be less than $2^k D_k(y'')$. We can express this sum as $c_1^2 + c_2^2 = c_1^2 + (2^k D_k(y'') - c_1)^2$. The minimum value for this parabola is $2^{2k-1} D_k(y'')^2$, when $c_1 = 2^{k-1} D_k(y'')$. Therefore we may write,

$$\begin{aligned}
\Pr[g = f_k^{y=y'} \mid z = z'] \geq & 2^{-k} \left(\prod_{x \in X \setminus \{x_1, x_2\}} D_k(f_k^{y=y'}(x)) \right) 2^{2k-1} D_k(f_k^{y=y'}(x_1)) D_k(f_k^{y=y'}(x_2)) \\
= & 2^{k-1} \left(\prod_{x \in X} D_k(f_k^{y=y'}(x)) \right)
\end{aligned}$$

Finally, plugging this into equation 4.2 we get the result

$$\Pr[z = z' | g = f_k^{y=y'}] \geq \frac{2^{-k}}{\prod_{x \in X} D_k(f_k^{y=y'}(x))} 2^{k-1} \left(\prod_{x \in X} D_k(f_k^{y=y'}(x)) \right).$$

Therefore we conclude that the oracle converter induced by S_2 has collision-conversion rate at least $\frac{1}{2}$.

4.3.3 Stronger lower bound for δ-min-k distributions

Note that following the same strategy, one can show a reduction of collision finding in an arbitrary min-k distribution D to collision finding in a δ-k-distribution. This is interesting because it affirms that the δ-k-distribution case is the most difficult out of all k-distributions. Clearly, if no elements in the support of D are associated with a probability mass less than $1/N$, the proof of Theorem 8 can be adapted by replacing all references of 2^{-k} as the probability of sampling each element from the flat distribution with a general probability $D(x)$, and replacing the general distribution D with a δ-k-distribution D_δ. The general case where D has elements associated with smaller probability mass than $1/N$ may be resolved by considering the distribution removing these elements and showing that it is computationally indistinguishable from the original.

In this section we give further evidence and establish an even stronger bound for finding collisions in the δ-k-distribution case.

Theorem 9. *For any q-query algorithm A,*

$$\Pr_{f \leftarrow D_{k,\delta}{}^X}[f(x) = f(x') : (x, x') \leftarrow A^f()] \leq O\left(\frac{q^2}{2^k} + \frac{q^3}{N}\right).$$

We give two proofs. The one presented here relies on a technique by Zhandry (Lemma 21). We give an alternative proof in Section 4.3.4 based on a reduction from an average version of a search problem which is hard to solve from the literature. This may serve as an intuitive explanation of the hardness of non-uniform collision finding. It also connects to the quantum algorithm we develop in Sect. 4.4.1 based on Grover's search algorithm.

Lemma 21. *[149, Theorem 7.2] Fix q, and let F_λ be a family of distributions on Y^X indexed by $\lambda \in [0, 1]$. Suppose there is an integer d such that for every $2q$ pairs $(x_i, y_i) \in X \times Y$, the function $p_\lambda := \Pr_{f \leftarrow F_\lambda}(f(x_i) = y_i, \forall i \in \{1, \ldots, 2q\})$ is a polynomial of degree at most d in λ. Then any quantum algorithm A making q queries can only distinguish F_λ from F_0 with probability at most $2\lambda d^2$.*

This lemma enables us to prove another lemma in turn.

Lemma 22. *For any q-query algorithm A,*

$$\left| \Pr_{f \leftarrow D_{k,\delta}{}^X}(A^f(\cdot) = 1) - \Pr_{f \leftarrow Y^X}(A^f(\cdot) = 1) \right| \leq 8q^2/2^k + 1/N.$$

Proof. For every $\lambda \in [0, 1]$, define D_λ on Y such that there is an element $m \in Y$ with $D_\lambda(m) = \lambda$ and for any $y \neq m$ $D_\lambda(y) = \frac{1-\lambda}{|Y|-1}$. Then define a family of distributions F_λ on Y^X where $F_\lambda := D_\lambda{}^X$, i.e., the output of each input is chosen independently according to D_λ.

For any $\{(x_i, y_i)\}_{i=1}^{2q}$, $p_\lambda := \Pr_{f \leftarrow F_\lambda}(f(x_i) = y_i, \forall i \in [2q]) = \lambda^t(\frac{1-\lambda}{|Y|-1})^{2q-t}$, where t is the number of occurrences of m in $\{y_i\}_{i=1}^{2q}$. Clearly p_λ is a polynomial in λ with degree at most $2q$.

Notice that $F_{2^{-k}}$ is exactly δ-min-k distribution $D_{k,\delta}$, and F_0 is uniformly random on $\hat{Y}^X$, where $\hat{Y} := Y \backslash \{m\}$. Therefore by Lemma 21,

$$\left| \Pr_{f \leftarrow {D_{k,\delta}}^X} (A^f(\cdot) = 1) - \Pr_{f \leftarrow \hat{Y}^X} (A^f(\cdot) = 1) \right| \leq 2(2q)^2 \cdot 2^{-k} = 8q^2/2^k \,.$$

Since Y^X and $\hat{Y}^X$ has statistical distance $\frac{1}{2}(N-1)(\frac{1}{N-1} - \frac{1}{N}) + \frac{1}{2}(\frac{1}{N} - 0) = 1/N$, we get that $\left| \Pr_{f \leftarrow {D_{k,\delta}}^X} (A^f(\cdot) = 1) - \Pr_{f \leftarrow Y^X} (A^f(\cdot) = 1) \right| \leq 8q^2/2^k + 1/N$. □

We are now ready to prove the stronger complexity for finding collision in a δ-min-k random function.

Proof of Theorem 9. Suppose that there is an A with

$$\Pr_{f \leftarrow {D_{k,\delta}}^X} [f(x) = f(x') : (x, x') \leftarrow A^f(\cdot)] = \varepsilon$$

using q queries. Then construct A' which on input oracle f, runs A and receives (x, x') from A. A' then output 1 if and only if $f(x) = f(x')$. By definition, we have that $\Pr_{f \leftarrow {D_{k,\delta}}^X} (A'^f(\cdot) = 1) = \varepsilon$. Meanwhile, note that A' makes $q + 2$ queries. Therefore by Zhandry's lower bound on finding collision in uniform random function (Lemma 17), we know that $\Pr_{f \leftarrow Y^X} (A'^f(\cdot) = 1) \leq O(\frac{(q+3)^3}{N})$. Then Proposition 22 implies that

$$\varepsilon \leq O(\frac{(q+3)^3}{N}) + 8(q+2)^2/2^k + 1/N = O(\frac{(q+2)^2}{2^k} + \frac{(q+3)^3}{N}) \,.$$

□

Corollary 4. *Any quantum algorithm needs* $\min\{2^{k/2}, N^{1/3}\}$ *queries to find a collision with constant probability. Specifically we need* $\Omega(N^{1/3})$ *if* $2^k \leq N < 2^{\frac{3k}{2}}$*, and* $\Omega(2^{k/2})$ *when* $N \geq 2^{\frac{3k}{2}}$*.*

4.3.4 Alternative proof of lower bound

Theorem 10. *Suppose* $|X| = M = o(\sqrt{N})$*. Any q-query quantum algorithm finds a collision in* $f \leftarrow X^{D_{k,\delta}}$ *with probability* $O(q^2/2^k)$*.*

We prove this by reducing a variant of Grover's search problem in [92] to finding a collision here. Define the following distribution E_λ on $\mathcal{F} : X \to \{0,1\}$: if $F \leftarrow E_\lambda$, then for any $x \in X$

$$F(x) = \begin{cases} 1 & \text{with prob. } \lambda\,; \\ 0 & \text{with prob. } 1-\lambda\,. \end{cases}$$

It has been shown in [92] that searching for a preimage of q in a function drawn according to E_λ is difficult. More precisely, for any quantum algorithm A making q queries, we define its success probability as

$$\mathsf{Succ}^{\lambda}_{A,q} := \Pr_{F \leftarrow E_\lambda}[F(x) = 1 : x \leftarrow A^F(\cdot)]\,.$$

Lemma 23. *([92, Theorem 1])* $\mathsf{Succ}^{\lambda}_{A,q} \le 8\lambda(q+1)^2$ *holds for any quantum algorithm A making at most q queries.*

Proof of Theorem 10. Let A be any quantum algorithm that makes at most q queries to $f \leftarrow {D_{k,\delta}}^X$ and finds a collision in f with probability ε. We show how to construct $\mathcal{B}$ that solves the above search problem for $\lambda = 1/2^k$ making $2q$ queries with probability $\varepsilon' = \varepsilon - \gamma$ and Lemma 23 then implies that $\varepsilon \le O(q^2/2^k)$.

$\mathcal{B}$ is given quantum access to $F \leftarrow F_\lambda$, and the mode m of $D_{k,\delta}$. Let $h : X \to Y \backslash \{m\}$ be a random function.[3] It simulates $\hat{f} : X \to Y$ which answers the queries from A:

$$\hat{f}(x) = \begin{cases} m & \text{if } F(x) = 1\,; \\ h(x) & \text{otherwise.} \end{cases}$$

After A has made q queries to $\hat{f}$, A outputs x and x'. $\mathcal{B}$ outputs one of them, e.g., x.

Note that $\mathcal{B}$ can implement each evaluation of $\hat{f}$ by two queries to F. Therefore $\mathcal{B}$ makes $2q$ queries to F at most. Next observe that $\hat{f}$ is distributed *identically* as $f \leftarrow X^{D_{k,\delta}}$, because $\Pr[\hat{f}(x) = m] = \Pr_{F \leftarrow F_\lambda}[F(x) = 1] = 1/2^k$ and the rest of $\hat{f}(x)$ is uniform over $Y \backslash \{m\}$. Therefore we know that $\hat{f}(x) = \hat{f}(x')$ with probability ε. Finally notice that when $M = o(\sqrt{N})$, h will be injective except with probability negligible in k. Therefore the collision only occurs at the mode, which implies that $F(x) = 1$ and B successfully finds a marked element in F. $\square$

[3]This can be efficiently simulated by a $2q$-wise independent hash function as justified by [150, Theorem 6.1] and [96, Lemma 2].

Corollary 5. *Any quantum algorithm needs $\Omega(2^{k/2})$ queries to find a collision in $X^{D_{k,\delta}}$ with constant probability even if the mode m of D is known, when $M = o(\sqrt{N})$.*

This result is basically subsumed by Theorem 9. When $N < 2^{2k}$, $\beta(D) = N$. Therefore $M = o(\sqrt{N}) = o(\sqrt{\beta(D)})$, and the function drawn is almost always injective. Hence the lower bound trivially holds. When $N \geq 2^{2k}$, Corollary 4 gives the same lower bound $2^{k/2}$.

The same proof strategy also works for general M, but then the probability that the collision occurs elsewhere other than the mode will introduce error, and it will match the bound we obtain from Theorem 9.

4.4 Upper bounds: (optimal) quantum algorithms

We derive a generic upper bound for finding collision in any min-k random functions. We adapt Ambainis's algorithm (Lemma 24) and describe a quantum algorithm NU-ColF (Figure 4.5).

Lemma 24. *([10, Theorem 3]) There exists a quantum algorithm ColF that makes $O(|X'|^{2/3})$ quantum queries to any function $f : X' \to Y$ with at least one collision that finds a collision with constant bounded error.*

Theorem 11. *Let $\beta := \beta(D_k)$. Let X be a set with $|X| = M = \Omega(\sqrt{\beta})$. Algorithm 4.5 NU-ColF finds a collision in $f \leftarrow X^{D_k}$ within $O(\beta^{1/3})$ queries with constant probability. Moreover with $O(k\beta^{1/3})$ queries the algorithm succeeds except with probability negligible in k.*

Proof. Since f is generated according to the min-k distribution, when restricting to any subset X_i, we can think of drawing each function value independently from D_k. Namely $f_i \sim {D_k}^{X_i}$ holds for all i. Therefore, by Lemma 2, we have that when $s \geq c\sqrt{\beta(D)}$ for some $c > 2$, f_i contains a collision with constant probability. If that is the case, Ambainis's algorithm will find a collision with constant probability using $O(|X_i|^{2/3}) = O(\beta(D)^{1/3})$ queries. We only need to repeat $t = O(k)$ times to succeed except with error negligible in k. □

Note that our algorithm NU-ColF is generic, and needs no additional information about D_k except for the value of β. By our characterization of $\beta(D_k)$ in Lemma 16, we obtain specific bounds for the two special distributions.

```
    Require: $f \leftarrow {D_k}^X$ as an oracle. Let $s, t$ be parameters to be specified later.
1 : Divide $X$ into subsets $X_i$ of equal size (ignoring the boundary case) $|X_i| = s$.
2 : Construct $f_i : X_i \to Y$ as the restriction of $f$ on $X_i$.
3 : for $i \in \{1, \ldots, t\}$ do
4 :   Run Ambanis's algorithm ColF on $f_i$ and get candidate collision $(x_i, x_i')$.
5 :   if $f(x_i) = f(x_i')$ then
6 :     return $(x_i, x_i')$
7 :   endif
8 : endfor
9 : return $\perp$
```

Figure 4.5: Collision Finding in Non-uniform Function.

Corollary 6. *There exists a quantum algorithm that finds a collision with constant probability using the following numbers of queries:*

- *flat-k:* $O(\beta^{1/3}) = O(2^{k/3})$ *and it is tight when* $M = \Omega(2^{k/2})$.
- *δ-min-k:* $O(\beta^{1/3}) = \begin{cases} O(N^{1/3}) & 2^k \leq N < 2^{2k}, \text{ tight when } N \leq 2^{3k/2} \\ O(2^{\frac{2k}{3}}) & N \geq 2^{2k} \end{cases}$

4.4.1 Quantum algorithm for min-k distribution with a mode known

We design an alternative collision finding algorithm (Algorithm 4.6), which performs slightly better in some settings. It is based on a version of Grover's algorithm [42, 83] for multiple marked items stated below.

Lemma 25. *Let* $f : X \to \{0, 1\}$ *be an oracle function and let* $Z_f = |\{x \in X : f(x) = 1\}|$. *Then there is a quantum algorithm* QSEARCH *using* q *queries that finds an* $x \in X$ *such that* $f(x) = 1$ *with success probability* $\Omega(q^2 \frac{Z_f}{|X|})$.

Theorem 12. *NU-ColF-Mode finds a collision using* $O(2^{k/2})$ *queries with constant probability.*

```
    Require: $f \leftarrow {D_k}^X$ as an oracle. A mode element $m$ of $D_k$.
1 : Run Grover's algorithm QSearch on $f$ to find $x$ with $f(x) = m$.
2 : Run Grover's algorithm QSearch on $f$ to find $x'$ with $f(x) = m$ and $x' \neq x$.
3 : if Either run of Grover's failed then
4 :   return $\perp$
5 : else
6 :   return $(x, x')$
7 : endif
```

Figure 4.6: Collision Finding in Non-uniform Function with a mode known.

Proof. Let $Z_f := |f^{-1}(m)|$. Let p_f be the probability that f is chosen, when drawn from ${D_k}^X$. Since we invoke QSearch twice, we find (x, x') with probability $\Omega\left((\frac{q^2 Z_f}{|X|})^2\right)$. Then algorithm NU-ColF-Mode succeeds with probability

$$\sum_f p_f \Omega\left(\frac{q^4}{M^2} Z_f^2\right) = \Omega\left(\frac{q^4}{M^2} \sum_f p_f Z_f^2\right) = \Omega(\frac{q^4}{M^2} \mathbb{E}[Z_f^2]) .$$

To compute $\mathbb{E}[Z_f^2]$, we define for every $x \in X$ an indicator variable $Z_x = \begin{cases} 1 & \text{if } f(x) = m; \\ 0 & \text{otherwise.} \end{cases}$, where $f \leftarrow {D_k}^X$, and clearly $Z_f = \sum_{x \in X} Z_x$. Since each output of x is drawn independently according to $D_{k,\delta}$, $\mathbb{E}[Z_x] = \varepsilon := 2^{-k}$ for all x, it follows that $\mathbb{E}[Z_x] = \mathbb{E}[Z_x^2] = \varepsilon$, and $\mathbb{E}[Z_x \cdot Z_{x'}] = \mathbb{E}[Z_x] \cdot \mathbb{E}[Z_{x'}] = \varepsilon^2$ for any $x \neq x'$ by independence. Therefore

$$\mathbb{E}[Z_f^2] = \sum_x \mathbb{E}[Z_x^2] + \sum_{x \neq x'} \mathbb{E}[Z_x Z_{x'}] = \Omega(M^2 \varepsilon^2) .$$

Hence the algorithm succeeds with probability $\Omega(q^4 \varepsilon^2) = \Omega((\frac{q^2}{2^k})^2)$. As a result, with $q = O(2^{k/2})$ many queries, we find a collision with constant probability. $\square$

Note that we still need $M = \Omega(2^k)$ to ensure there is a collision on m in f. When $N \geq 2^{3k/2}$, Theorem 12 gives a better bound $(2^{k/2})$ than Theorem 11 ($N^{1/3}$ when $2^{3k/2} \leq N < 2^{2k}$ and $2^{2k/3}$ when $N \geq 2^{2k}$).

```
    Require: Let $f \leftarrow {D_{k,\flat}}^X$ be a random function whose images are sampled according to $D_{k,\flat}$ on a set $Y$. Let $D'$ be a distribution on a set $Y'$ with min-entropy $k$. Let $S_2$ be the $D_{k,\flat} \to D'$ redistribution function sampler from Algorithm 4.3. Let $\mathcal{A}$ be an adversary for (second-)preimage-finding in $D'$. Let $y$ be the preimage target, or let $x$ be the second-preimage target.
1 : Run $S_2$ and store its output as $r$. Implement an oracle for $r$.
2 : Construct an oracle converter $\mathcal{C}$ using the oracles for $f$ and $r$. The responses of $\mathcal{C}$ are now distributed according to $D'$. Refer to the function implemented by $\mathcal{C}$ as $g$.
3 : For the second-preimage problem, provide $x$ as the second-preimage target. For the preimage problem, sample a uniform $x \in X$ and provide $y' = r(x, y)$ as the target.
4 : Initialize $\mathcal{A}$. For each query made by $\mathcal{A}$, forward the query to $\mathcal{C}$ and return the response to $\mathcal{A}$.
5 : When $\mathcal{A}$ returns a (second-)preimage $x'$ in $g$, return $x'$.
```

Figure 4.7: Preimage converter via oracle converter

4.5 Preimage- and second-preimage resistance of non-uniform random functions

In this section we sketch out a proof that the same redistribution function sampler S_2 from Algorithm 4.3 can be used to demonstrate similar results to Theorem 8, but relating to (second) preimage-resistance. First we give a formal definition for the problem of finding a (Second) Preimage:

Definition 24 ((Second-)Preimage finding problem)**.** Let $f \leftarrow D^X$ be a function of min-entropy k. For any $y \in Y$, we call $x \in X$ a *preimage* in f of y if $f(x) = y$. We refer to the problem of finding a preimage of y which has been generated by sampling a uniformly random $x \in X$ and computing $y = f(x)$ as the *preimage finding problem* in D. We refer to the problem of being given a uniformly sampled $x \in X$ and finding a $x' \in X$ such that $x \neq x'$ and $f(x') = f(x)$ as the *second-preimage finding problem* in D.

The reduction algorithm will then be similar to that of Figure 4.1, but specified to the problem of (second-)preimage resistance, and using Redistribution Function Sampler S_2 from Figure 4.3.

We then show a similar lemma to that of Lemma 19.

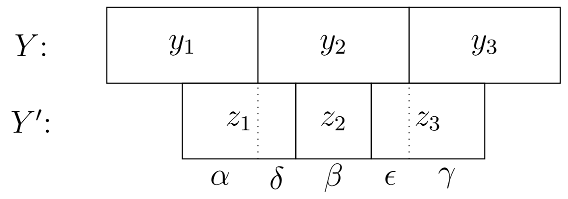


Figure 4.8: Visualizing how distributions line up for preimage problems.

Lemma 26. *Suppose there exists an algorithm $\mathcal{A}$ which solves (second-)preimage finding in a distribution D' with probability at least $P_{\mathcal{A}}$, using q queries to an oracle for a function g whose responses are distributed according to D'. Then the process in Figure 4.7 initialized with S_2 and $\mathcal{A}$, denoted $\mathcal{R}_{S_2,A}$, solves (second-)preimage finding in $D_{k,\flat}$ with probability at least $P_{\mathcal{A}}/2$ using $2q$ queries to an oracle for f whose images are distributed according to $D_{k,\flat}$.*

Proof. We have already shown that the function g presented to $\mathcal{A}$ in the algorithm in 4.7 will exactly have the distribution D'. Next we need to ensure that the target for the (second-)preimage problem has the same distribution. For the second-preimage problem, this can be seen to be the case, because the target is simply a uniformly random $x \in X$, as required.

For the preimage problem things are slightly more complex. Only the point y is provided to $\mathcal{R}_{S_2,A}$, and so to sample a point $y' \in Y'$, we choose a uniformly random point $x \in X$ and set $y' = r(x, y)$. We must show that the distribution of y' is exactly D'. Note that this is a stronger requirement than what is guaranteed by Definition 21, which only states that the distribution of $r(x, f(x))$ is exactly D', and does not specify what happens for $r(x, y)$ when $y \neq f(x)$. However, S_2 does in fact satisfy this stronger property, because r is constructed for each $(x, y) \in X \times Y$, not just $(x, f(x))$ for $x \in X$.

So our remaining task is to show that $\mathcal{R}_{S_2,A}$ succeeds with probability at least $P_{\mathcal{A}}/2$.

At the end of the execution of $\mathcal{R}_{S_2,A}$, with probability $P_{\mathcal{A}}$, $\mathcal{A}$ produces a $x' \in X$ such that $g(x') = r(x', f(x')) = y' = f(x, y)$. For the second preimage problem there is the further restriction that $x' \neq x$ and that $y = f(x)$.

We then have solved the (second-)preimage problem, as long as $f(x') = y$. So we need to consider the probability that $f(x') = y$, given that $r(x', f(x')) = r(x, y)$.

We show that with probability at least $1/2$, it is the case that $f(x') = y$. Consider the following case. The 'target point' y that we are trying to find a (second) preimage to is y_2.

Our redistribution function, with the second half of the output being y_2, may return many possible outputs. For simplicity, and without loss of generality, we consider the case where there are three possible outputs: z_1, z_2, or z_3. We will explain in more detail in a moment why we may do this without loss in generality. While $z_2 = r(a, b)$ means $b = y_2$, if $r(a, b) = z_1$ then $b = y_1$ or y_2, and if $r(a, b) = z_3$ then $b = y_2$ or y_3.

We define the following probabilities, taken over uniformly random a and b (visualized in Figure 4.8):

$$\delta_i := \Pr[r(a,b) = z_i \wedge b = y_2], \quad i = 1, 2, 3\,;$$
$$\alpha := \Pr[r(a,b) = z_1 \wedge b = y_1], \quad \beta := \Pr[r(a,b) = z_3 \wedge b = y_3]\,.$$

Then we can perform the calculation of the probability that $f(x') = y_2$ (conditioned on the fact that $y = y_2$) as follows:

$$\begin{aligned}
\Pr[f(x') = y_2] =& \Pr[f(x') = y_2 \wedge y' = z_1] + \Pr[f(x') = y_2 \wedge y' = z_2] \\
&+ \Pr[f(x') = y_2 \wedge y' = z_3] \\
=& \Pr[y' = z_1] \Pr[f(x') = y_2 | y' = z_1] \\
&+ \Pr[y' = z_2] \Pr[f(x') = y_2 | y' = z_2] \\
&+ \Pr[y' = z_3] \Pr[f(x') = y_2 | y' = z_3] \\
=& 2^k \delta_1 \frac{\delta_1}{\alpha + \delta_1} + 2^k \delta_2 + 2^k \delta_3 \frac{\delta_3}{\beta + \delta_3} \\
=& 2^k \left(\delta_2 + \frac{\delta_1^2}{\alpha + \delta_1} + \frac{\delta_3^2}{\delta_3 + \beta} \right).
\end{aligned}$$

We can see that we additionally constrained by the facts that $\alpha + \delta_1 \leq \frac{1}{2^k}$ and $\beta + \delta_3 \leq \frac{1}{2^k}$. So we can see that this equation is minimized by setting: $\delta_2 = 0$, $\alpha = \beta = \delta_1 = \delta_3 = \frac{1}{2^{k+1}}$. In this case we get

$$2^k \left(0 + \frac{1/2^{2k+2}}{1/2^k} + \frac{1/2^{2k+2}}{1/2^k} \right) = 1/2.$$

The reason that this is without loss of generality is because the equation was minimized by having $\beta = 0$. In other words, the probability of finding a correct second preimage was minimized when there was *no* $z \in Y'$ such that $f(a, b) = z$ guaranteed that $b = y$. Then because there are at most two $z \in Y'$ such that there are multiple $b \in Y$ with $f(a, b) = z$, we have that this case with $\beta = 0$ does indeed minimize the probability of recovering a (second-)preimage. □

Using this lemma, we can then compose and prove a formulation of Theorem 8, but with respect to (second-)preimage resistance.

Theorem 13. *Let $f_D \leftarrow D^X$ be a random function whose outputs are chosen independently according to a distribution D of min-entropy k. Then any quantum algorithm making q queries to f_D solves the (second-)preimage problem with probability at most* $16 \cdot 2^{-k}(2q + 1)^2$

Proof. We have shown that an adversary $\mathcal{A}$ that can solve the (second-)preimage problem on a distribution D in q queries with probability $P_{\mathcal{A}}$ implies the existence of a way to solve the (second-)preimage problem on a flat distribution function $f_{D_{k,\flat}}$ in $2q$ queries with probability $P_{\mathcal{A}}/2$.

We can then reduce the hard average-case search problem in Lemma 23, to finding preimiage and second-preimage in a flat distribution, following similar analysis for uniform random functions as in [92]. As a result, any adversary's success is at most $8 \cdot 2^{-k}(q + 1)^2$ in q queries.

We have shown that if a quantum algorithm exists that can solve the (second-) preimage problem on f_D in q queries with probability $P_{\mathcal{A}}$, then there exists an algorithm that can solve (second-)preimage resistance on a flat distribution in $2q$ queries with probability $P_{\mathcal{A}}/2$. This tells us that

$$P_{\mathcal{A}}/2 \leq 8 \cdot 2^{-k}(2q + 1)^2,$$

from which the result follows. □

Next, we apply (generalized) Grover's search algorithm to solve the preimage and second-preimage problems in min-k distributions.

Theorem 14. *Let $\beta := \beta(D_k)$. Let X be a set with size $\Omega(\beta)$. Then in q quantum queries the* QSearch *algorithm referred to in Lemma 25 solves (second) preimage problems with probability $\Omega(q^2/\beta)$.*

Proof. Given $x \leftarrow X, f \leftarrow {D_k}^X$, let $Z_{f,x} := \left|\{z \in X : f(z) = f(x)\}\right|$ be the size of the preimage of $f(x)$. By Lemma 25, QSEARCH will succeed in finding a preimage of $f(x)$ with probability

$$\sum_{f,x} p_{f,x} \Omega(q^2 \frac{Z_{f,x}}{|X|}) = q^2 \frac{1}{|X|} \mathbb{E}_{f,x}(Z_{f,x})\,.$$

We can estimate $\mathbb{E}_{f,x}(Z_{f,x}) = |X| \cdot \sum_y D_k(y)^2 = |X|/\beta$. Therefore, we find a preimage with success probability $\Omega(q^2/\beta)n$. The same analysis works for finding a second preimage.

$\square$

Chapter 5

Instantiating the Quantum Random Oracle

5.1 Introduction

In this chapter, we show that there exist digital signature schemes that can be proven secure against any poly-time quantum adversaries in the *quantum random-oracle model* [40], but they can be broken by a *classical* poly-time adversary when the random oracle is instantiated by any poly-time computable hash function family. This extends to the quantum setting the impossibility of instantiating a classical random oracle [21, 46, 47, 81, 113, 119].

Given the classical result (e.g., [46]) that there exists a secure signature scheme in the random oracle model but insecure under any efficient instantiation, the first doubt to clear up is probably why it does not immediately follow that a *quantum* random oracle cannot be instantiated as well. The reason is that the signature scheme in the classical result may as well get *broken* in the quantum random oracle model. In other words, all one needs to do is to prove quantum security of these classical constructions in the quantum random oracle model. This is exactly what this work does: we show that three examples in the classical setting [46, 47, 113] can be proven secure in the quantum random oracle model, and hence they demonstrate that the quantum random oracle model is *unsound* in general.

We dive into an overview of the proofs right away, so that those who are familiar with this subject can quickly digest the gist and walk away satisfied (or disappointed). If you are a more patient reader, you can come back here after enjoying the (more conventional) introduction.

Let us first review the classical examples [46, 47, 113] to be analyzed in the quantum random oracle model, and we present them under a unified framework which we hope will be easy to grasp. They all start with a secure signature scheme Σ and a function F, and Σ is "punctured" so that the signing algorithm would simply reveal the signing key when the function F is "non-random" (e.g., instantiated by a concrete hash function). To break it, an adversary just needs to convince the signing algorithm that F is indeed non-random. Therefore, it boils down to designing a proof system where a prover (adversary in the signature setting) proves "non-randomness" of a given function to a verifier (signing algorithm); whereas if the function is indeed random, no prover can fool the verifier to accept. The natural approach to such a proof system is based off the intuition that it is difficult to *predict* the output of a random oracle on an unknown input. The three classical examples nurture this intuition in two variations: predicting on a *single* input or *multiple* inputs.

1. The basic idea in [46] is to have the prover provide *an* input where the output is predictable and can be efficiently verified by the verifier. For starters, suppose we want to rule out a specific hash function H, then the prover can pick an arbitrary x and the verifier just checks if $F(x) = H(x)$. The verifier always accepts when F is instantiated by H, but accepts only with negligible probability if F is random. This immediately implies that for any function family, in particular the family of *poly-time computable* functions $\mathcal{H} = \{\mathcal{H}_n = \{H_s\}_{s\in\{0,1\}^n}\}$[1], we can construct a signature scheme following the idea above, where a (random) member in $\mathcal{H}$ is chosen as implementation of F, and the signing algorithm reveals the signing key whenever the "non-randomness" verification passes. Note that, nonetheless, the construction depends on the function family, which is *weaker* than the goal of establishing a signature scheme that is secure in the random oracle model, but insecure when implemented with a function from the function family $\mathcal{H}$.

   Diagonalization comes in handy to reverse the quantifiers. The prover will provide a description s of a function H_s, which purportedly describes the function F. Then the verifier runs H_s on s and checks if it matches $F(s)$. Clearly, when F is implemented by a member $H_s \in \mathcal{H}$, the description s is public (i.e., part of the verification key), and it is trivial for the prover to convince the verifier. Nonetheless, if F is a random oracle $\mathcal{O}$, the event $\mathcal{O}(s) = H_s(s)$ occurs only with negligible probability for any s that a prover might provide.

[1]We assume a canonical encoding of functions into binary strings, under which s is the description of a function. Complexity is measured under security parameter n.

A technicality arises though due to the time complexity for computing $H_s(s)$ for all $\mathcal{H}$. Loosely speaking, we need a universal machine for the family $\mathcal{H}$ that on a description s computes $H_s(\cdot)$. Such a machine exists, but would require slightly super-polynomial time, which makes the verifier (i.e., signing algorithm) inefficient. This final piece of the puzzle is filled by CS-proofs introduced by Micali [116]. A CS-proof allows verifying the computation of a machine M, where the verifier spends significantly less time than the time to run M directly. This naturally applies to the problem here. Instead of running the universal machine to check $H_s(s) = F(s)$ by the verifier, the prover generates a CS-proof on the input $\langle M, s\rangle^F$ (relative to F) certifying the statement $M(s) := H_s(s) = F(s)$, which the verifier can check in poly-time. When $F = \mathcal{O}$ is a random oracle, $\langle M, s\rangle^{\mathcal{O}}$ (relative to $\mathcal{O}$) is almost always a false statement, and the soundness of the CS-proof ensures that verifier will reject with high probability. Micali proved in general the soundness of CS-proofs in the random oracle model (to avoid confusion, in CS-proofs think of a random oracle independent of F).

2. Another strategy for proving "non-randomness", as employed in Maurer *et al.* [113] and Canetti *et al.* [47], is to predict on *multiple* inputs. This offers a direct *information-theoretical* analysis without relying on CS-proofs.

   In essence, a prover provides a machine π that allegedly predicts the output of F on sufficiently many inputs, and the verifier can run π and compare with the answers from F. This is easy for the prover when F is instantiated by $\mathcal{F}$ where the description s is given. On the other hand, by tuning the parameters, a counting argument would show that the randomness in a random oracle is overwhelming for any single machine (even inefficient ones!) to predict. Specifically, the "predicting" machine π needs to match with F on $q = 2|\pi| + n$ inputs (i.e., the number of correct predictions has to be significantly more than the length of the description of the machine). Suppose that F is a random oracle $\mathcal{O} \leftarrow \{f : \{0,1\}^* \to \{0,1\}\}$ that outputs one bit (for the sake of simplicity), then for any π the probability that it will match $\mathcal{O}$ on q inputs is at most $2^{-(2|\pi|+n)}$. A union bound on all machines of length n shows that p_n, the probability that some length-n machine is a good predictor, is at most $2^n \cdot 2^{-(2n+n)} = 2^{-n-k}$. Another union bound shows that regardless of their length, no machine can be a good predictor, since $p := \sum_{n=1}^{\infty} p_n = 2^{-n} \sum_n 2^{-n} \leq 2^{-n-1}$ is negligible. To get around the issue of the run-time of the machine π, the prover also has to provide an upper bound t on the run-time of π encoded in a unary representation. If π fails to complete after t steps, then a random output is returned. In this way, the runtime of signing procedure is guaranteed to be polynomial in the size of the input. However, this naturally implies that the signatures to be signed are quite long.

3. Both examples above suffer from an artifact. Namely the signature schemes need to be able to sign *long* messages or otherwise maintain *states* of prior signatures. This is rectified in [47], where a *stateless* scheme that signs only messages of polylogarithmic length is proven secure in the random oracle model but insecure under any instantiation.

   At the core of this construction is an *interactive* counterpart of the *non-interactive* proof system in part 2 above. It can be viewed as a *memory* delegation protocol, where a verifier with limited (e.g., poly-logarithmic) memory wants to check if the machine provided by the prover is a good predictor. Roughly speaking, it will execute the machine step by step and use the prover to bookkeep intermediate configurations of the machine. However, the configurations may be too long for the verifier to store and transit to the prover. Instead, the verifier employs a Merkle tree and only communicates an *authentication path* of the configuration with the prover. In particular, the verifier will memorize only a secret authentication key in between subsequent rounds. The security of the "punctured" signature scheme reduces to essentially a stronger *unforgeability* of a valid authentication path in a Merkle tree with respect to a random oracle, which is proven classically.

All of these examples rely on a Turing machine representation of the description of the function F. The fact that we cannot a-priori determine the run time of F when presented with this description is what leads to requiring complications such as CS-proofs or stating the run-time in unary. Different ways to describe F can lead to different consequences however. The circuit representation of a function means that the run time is inherently embedded in its description. While this solves the issue of ensuring the signing protocol runs in polynomial time in its input, it denies the function the ability to contain an arbitrary loop, and so we focus on a Turing machine representation.

Proving security of separation examples in QRO. Once the constructions and classical analysis are laid out, proving their security in the quantum random oracle model becomes more or less mechanic, given the techniques developed for QRO so far [12, 13, 68, 141, 149].

1. (Example in [46] with CS-proofs.) Following the classical proof, we first show that the quantum security reduces to one of three cases: 1) hardness of a Grover-type search problem, which ensures that an adversary cannot feed the CS-proof a true statement in the case of a random oracle; 2) security of the original signature scheme; and 3)

soundness of CS-proofs against quantum adversaries. A precise query lower bound for the search problem follows by standard techniques. And thanks to a recent work [53], CS-proofs are proven sound against quantum adversaries.

2. (Example in [47, 113] based on information-theoretical analysis.) It is easy to verify that the information-theoretical argument sketched above holds regardless of the kind of adversaries, and as a result the "punctured" signature scheme remains secure in the quantum random oracle model (and against quantum adversaries).

3. (Example in [47] that only needs to sign short messages.) Our proof follows the classical one, where we first carefully verify and lift the reduction to the (stronger) unforgeability of Merkle tree against quantum adversaries, and then prove this property in the quantum random oracle model.

   Specifically, we can model the unforgeability game as follows. Think of two correlated random oracles $\mathcal{O} : \{0,1\}^* \to \{0,1\}^{\ell(n)}$ and $\mathcal{O}' := \mathcal{O}(ak, \cdot)$ where ak is a random authentication key kept secret. Given *quantum* access to $\mathcal{O}$ and *classical* access to $\mathcal{O}'$, the adversary needs to come up with an authentication path $(\langle \sigma_1, \ldots, \sigma_d \rangle, \langle (v_{1,0}, v_{1,1}), \ldots, (v_{d,0}, v_{d,1}) \rangle, t)$ where $\sigma_i \in \{0,1\}$, $t = \mathcal{O}'(d, \mathcal{O}(0, v_{1,0}, v_{1,1}))$ and $v_{i,\sigma_i} = \mathcal{O}(i, v_{i+1,0}, v_{i+1,1})$ for every $i = 1, \ldots, d-1$. We prove that this is infeasible by reductions from a randomized decisional search problem and collision finding in random functions [42, 92, 150].

Background and motivation. The random oracle model, since its introduction [24], has proven a popular methodology for designing cryptographic schemes. Basically a construction is first described and analyzed in an idealized setting where a random function is available as a black-box. To implement it in the real-world, one substitutes a cryptographic hash function for the random oracle. This methodology often leads to much more efficient schemes than alternatives. Examples include digital signatures by the Fiat–Shamir transform [74], hybrid public-key encryption following Fujisaki–Okamoto-type transforms [25], as well as succinct non-interactive zero-knowledge arguments that rise with the trending technology of blockchain and cryptocurrencies [28]. Its popularity is also attributed to the fact that one can often prove security in the random oracle model which is otherwise much more challenging or simply unknown.

It is, however, exactly the latter advantage that stirred considerable debate. *What does a security proof in the random oracle model mean?* To be pragmatic, a random-oracle proof at least serves as a sanity check that rules out inherent design flaws. Indeed, in practice most constructions that are instantiated from ones proven secure in the random oracle model have stood up to extensive cryptanalysis. More formal pursuit, however, arrives at an irritating message. There are separation examples which show secure constructions in the random oracle model, but will be trivially broken whatever "nice" functions we use to instantiate the random oracle. Namely, the methodology is *unsound* in general. This does not mean all schemes following this approach are insecure. In fact, some random-oracle scheme can be instantiated under strong but reasonable assumptions and achieve desirable security in the real-world [104]. To say the least, a question mark lingers on schemes developed under this methodology.

Quantum computing adds another layer of complication to the issue here (and the overall landscape of cryptography). Because of the threats to widely deployed cryptosystems [132], a growing effort is undertaken to design and transition to so called *post-quantum cryptography* – a new set of cryptosystems that hopefully resist quantum attacks. In particular, the random oracle model has been re-examined in the presence of quantum adversaries. Since eventually a scheme (designed in RO) will be realized via a cryptographic function, whose specification is known in public, a quantum adversary can in principle construct a coherent quantum circuit that evaluates the hash function in quantum *superposition*. Consequently, when analyzing the scheme in the idealized setting, it seems necessary to grant quantum superposition queries to the random oracle by a quantum adversary. This brings about the quantum random oracle model [40]. The rationale is, very informally, good cryptographic functions are lacking structures for a quantum computer to exploit (aside from generic speedup due to quantum search), and hence realizing a scheme proven secure relative to a quantum random oracle this way is a fine practice.

Formally analyzing security in the quantum random oracle model turns out to be challenging. Many classical proof techniques, such as simulating and programming a random oracle on-the-fly or recording the queries, seem to fail due to unique features of quantum information. Thanks to a lot of continued effort, in recent years, researchers managed to develop various techniques for reasoning about the quantum random oracle model, and restored the security of many important constructions against quantum adversaries [13, 64, 68, 89, 92, 141, 150, 151]. This just adds more at stake regarding "what does it mean that a scheme is proven secure in the *quantum* random oracle model?"

How to interpret this result? Our work shows that in general, security in the quantum random oracle model could be vacuous in a real-world implementation. There seems a dilemma, probably more puzzling than the classical situation. On the one hand, since a quantum adversary is given more power (e.g., quantum computation and superposition access), security in the quantum random oracle provides more justification that the construction is solid. And this indeed explains the difficulty in establishing security in the quantum random oracle. Hence it might occur that security in QRO would be sufficiently strong to imply security in the plain model and rule out separations. Our work nonetheless shows otherwise, and it reveals the other side of the dilemma. Any bit of success in restoring proof techniques in the quantum random oracle model just casts another bit of shadow on this methodology, since the seemingly stronger quantum security does not promise the security of real-world implementations, not even their security against *classical* adversaries only.

On the other hand, many cryptosystems that have been proven secure in the random oracle model have fared well in retrospect [105]. The use of the random oracle model to get a proof can allow for schemes that are simpler and more efficient than those in the standard model. While proofs in the quantum random oracle model appear more difficult, every year new techniques, more general and user-friendly, are developed to establish quantum random oracle model security [64, 87, 92, 151]. This has led researchers to question what guarantees security in the quantum random oracle provides versus the classical random oracle model. In this line, our results can be taken as further justification that the difference between these models does not appear to be a large one. If one hopes to show that security in the classical and quantum oracle model provide a similar set of assurances, then it seems natural that the same instantiability problems exist in the classical random oracle model as well as the quantum counterpart.

5.2 Chapter Background

5.2.1 The (Quantum) Random Oracle Model and Notation

The random oracle model, originally devised in [24], replaces a cryptographic hash function with an entirely random oracle. The reduction algorithm is often allowed to manage this oracle, and can perform operations like looking up the queries that the adversary makes to it, or programming the oracle on inputs of interest. Using the random oracle model can often greatly simplify a proof or even enable a proof where otherwise not known or possible.

The intuitive idea behind the soundness of the random oracle methodology is that an adversary interacting with a scheme is unlikely to take advantage of the structure of the hash function. For most cryptographic schemes, even the adversary is likely to treat the hash function as a 'black box', and so by treating it as such, we can derive proofs for schemes that otherwise may not exist.

However, as we discussed in the previous chapters, a full consideration of a quantum adversary means that we should allow superposition queries to the random oracle. In the quantum random oracle model, the reduction algorithm still manages the oracle, but now the adversary must be allowed to make a superposition query to this oracle. For an oracle $\mathcal{O} : \mathcal{D} \to \mathcal{R}$, the reduction provides access to a unitary $U_\mathcal{O}$ which performs the action

$$U_\mathcal{O} : \sum_{x \in \mathcal{D}, y \in \mathcal{R}} \alpha_{x,y} |x\rangle|y\rangle \mapsto \sum_{x \in \mathcal{D}, y \in \mathcal{R}} \alpha_{x,y} |x\rangle|y \oplus \mathcal{O}(x)\rangle,$$

where the input must be a valid quantum state, i.e., the sum of the square amplitudes of the $\alpha_{x,y}$'s must be 1.

For clarity, we will denote a random oracle as $\mathcal{O}$, while actual instantiations of random oracles (e.g., typically hash functions) are denoted H. When describing a scheme where a function may replaced with a random oracle in the proof, or with a hash function in the real world, we will denote this function with F. The security parameter of a scheme is denoted by λ, while the output length of a hash function is denoted n. While in general these are two separate but related parameters, throughout this work it is the case that $\lambda = n$, and we do not distinguish between the two.

5.2.2 Computationally Sound Proofs

Computationally sound proofs, introduced by Micali in 2000 [116], allow extremely efficient verification of a problem $\mathcal{L}$ with the help of a prover. In our context, CS-proofs are useful for showing the validity of a computation without having to run the computation. Imagine a description of an arbitrary function f, which may take super-polynomial time to run on an input x, but will result in $f(x) = y$. A CS-proof system allows us to generate a proof π that $f(x) = y$. Even though f may take a super-polynomial amount of time to run, the CS-Proof verification system allows a verifier, on input of f, x, y, and π to verify that $f(x) = y$ in only poly-logarithmic time.

For concreteness in our work, a CS-proof system consists of two algorithms: CSProve and CSVerify. Both algorithms implicitly take a security parameter n. CSProve also takes in a function f and an input x, and returns a value y and a proof π. The CSVerify function takes in a function f, an input x, an output y, and a proof π. It returns either accept or reject, based on the validity of the proof. Crucially, the CSVerify function runs in time poly-log in the security parameter n, and not in relation to the time it takes f to run.

The correctness property states that for an honestly generated proof π, the CSVerify function will always accept. The soundness property ensures that if $f(x) \neq y$, then it is computationally infeasible to find a proof π that will cause $\mathsf{CSVerify}(f, x, y, \pi)$ to return accept. The soundness of CS-proofs was originally shown in the random oracle model. Very recently, Chiesa et al. [53], proved the soundness of CS-proofs in the quantum random oracle model, which we will rely on in this work.

In this work, while the function f is arbitrary, it is not allowed to depend on calls to external oracles, and in particular, cannot depend on the random oracle. This is intended to reflect the fact that f is meant to be an instantiation of the random oracle, and thus should not depend upon it. However, recently this approach has been questioned. In a recent paper by Zhandry [152], the author considers what happens to model separations like these when f can depend on the random oracle.

5.3 Instantiating a Quantum Random Oracle

In this section we define three signature schemes, Σ_1, Σ_2, and Σ_3, such that:

- Σ_1 is secure in the quantum random oracle model, but insecure if that random oracle is instantiated with some specific hash function H.
- Σ_2 is secure in the QROM, but insecure when the random oracle is instantiated with any of a pre-defined set of hash functions $\{H_1, \ldots, H_m\}$.
- Σ_3 is secure in the QROM, but insecure if the random oracle is ever instantiated with *any* polynomial-time function.

These signature schemes lift the results in [46] to the quantum random oracle model. In all cases, the only assumption we require is that we have a signature scheme $\Sigma_0 = (\mathsf{KeyGen}_0, \mathsf{Sign}_0, \mathsf{Verify}_0)$ which is existentially unforgeable in the quantum random oracle model. Examples of schemes secure in the quantum random oracle model with no additional assumptions include the stateful LMS signatures [66] and the stateless SPHINCS+ framework [31] (both hash-based signatures). If one is willing to accept a computational assumption such as ring-LWE, many other signature schemes, including several of those under consideration in the NIST standardization process serve as examples [103]. It is a notable complication of Σ_0 is that we require it to be secure against a *slightly super-polynomial* adversary. It must not be compromised by an adversary running in time $n^{\log(n)}$. Security bounds for generic attacks against SPHINCS+ (relevant for the quantum random oracle model) require on the order of $\sqrt{2^n}$ quantum hash function evaluations in order to compromise the security with constant probability. Thus by setting Σ_0 as SPHINCS+, we satisfy all requirements.

```
$\mathsf{KeyGen}_1(1^\lambda)$
1 : $(pk, sk) \leftarrow \mathsf{KeyGen}_0(1^\lambda)$
2 : return $(pk, sk)$

$\mathsf{Verify}(pk, msg, \sigma_1)$
1 : parse $\sigma_1 = \sigma_0 || x$
2 : return $\mathsf{Verify}_0(pk, msg, \sigma_0)$

$\mathsf{Sign}_1(sk, msg)$
1 : $\sigma_0 \leftarrow \mathsf{Sign}_0(sk, msg)$
2 : if $F(msg) = H(msg)$ then
3 :     return $\sigma_1 = \sigma_0 || sk$
4 : else
5 :     return $\sigma_1 = \sigma_0 || 0^{l_{sk}}$
6 : endif
```

Figure 5.1: Signature scheme Σ_1

5.3.1 Warm up — schemes Σ_1 and Σ_2

The first step in considering the instantiation of a random oracle is to consider instantiation with a single hash function, H. Then we can define the scheme Σ_1 in Figure 5.1. Clearly this scheme satisfies the correctness property, as Σ_0 does.

This scheme is existentially unforgeable under chosen message attack (eu-cma) secure in the (quantum) random oracle model, where F is replaced with an oracle $\mathcal{O}$. This is intuitively because in this case, the security reduces to that of Σ_0 unless an adversary is able to find a msg such that $\mathcal{O}(msg) = H(msg)$ (which occurs for every possible input with uniform and independent probability $1/2^n$).

Furthermore, this scheme is *insecure* if it is instantiated with H replacing the random oracle. Then the adversary is able to trivially break security, as the condition $H(msg) = H(msg)$ is always satisfied and $\sigma_1 = \sigma_0 || sk$ will be returned for any message.

The next step is considering a finite collection of m hash functions, $\mathcal{H} = \{H_1, H_2 \ldots, H_m\}$. Then we can define Σ_2 similarly to Σ_1, but change the condition to first check if $msg \in \{1, \ldots, m\}$ (in some encoding of the integers 1 through m) and if so, further check if $F(msg) = H_{msg}(msg)$.

The analysis in the (quantum) random oracle model is again fairly straightforward. For any random oracle $\mathcal{O}$, the probability that $\mathcal{O}(i)$ matches $H_i(i)$ is $1/2^n$, and so a straightforward union bound establishes that the probability $\mathcal{O}(i)$ matches $H_i(i)$ for any i in $\{1, \ldots, m\}$ is at most $\frac{m}{2^n}$. When m is small (e.g., polynomially sized in λ), this is small enough that it is likely to not be possible that an adversary can make a query that provides them with sk. Even for a large m, each $i \in \{1, \ldots, m\}$ will have the property that $\mathcal{O}$ and H_i match with probability $1/2^n$, and so an adversary must perform an unstructured search to find such an i. By the optimality of Grover's algorithm [43], an adversary's ability to break Σ_2 in the (quantum) random oracle model reduces to their ability to break Σ_0 unless the adversary makes a subexponential number of queries to the hash function.

However, as before, if F is actually replaced by any one of the H_i's, an adversary can easily break the scheme by querying i to the signing oracle.

5.3.2 Signature scheme Σ_3

Schemes Σ_1 and Σ_2 are only to gain an intuition for the full result, Σ_3, which is a signature scheme that is secure in the quantum random oracle model, but insecure when the oracle is instantiated with any polynomial-time function as the hash function. Following the strategy for Σ_2, we would like to fix some enumeration of all algorithms that one may use as a hash function, say $\mathcal{H} = \{H_1, H_2, \dots\}$, with $H_i : \{0,1\}^* \to \{0,1\}^n$. Then as before, we would modify an eu-cma secure scheme Σ_0 to introduce a check in the signing algorithm to interpret *msg* as a non-negative integer, and check if $F(msg) = H_{msg}(msg)$. However, there are several issues that must be resolved to make this fully rigorous. Such a set of functions cannot simply be defined and used in the signature scheme, as the signature scheme requires that on input i, hash function H_i is actually run.

To fix this, we start with an enumeration of *all* algorithms, $\mathcal{A} = \{A_1, A_2, A_3, \dots\}$. We make no assumptions about this enumeration except that we can efficiently swap between the index i and some standard description of A_i (as a Turing machine). Changing between A_i and i should not be seen as a computational task to carry out, but rather a reinterpretation of the same data. Algorithms are encoded, using some standard encoding depending on the computational model, into bit strings, which can then easily be interpreted as an integer. To think of a construction that achieves this, it is helpful to think of quantum circuits. If we are working with l registers, then we can interpret the index i as a value in $\{0,1\}^*$ which specifies which gates are applied to which registers in what order. From a description of a quantum circuit, it is easy to convert this into a binary string, and then an index, and vice versa. To be reversible and match the format of a hash function, we can then consider all circuits that perform the mapping $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus A_i(x)\rangle$.

Note that not all of these algorithms necessarily run in polynomial-time in the security parameter. It is of course impossible to tell which algorithms will even terminate. We would like to assume that when a random oracle is instantiated, the function it is instantiated with will run in polynomial time in the security parameter. As well, these algorithms do not necessarily have the correct output length of n bits.

To fix this, we modify each algorithm in the following way: For each algorithm, stop after taking $n^{\log n}$ steps, and pad or truncate the output (in an arbitrary way) so that each

```
KeyGen_3(1^λ)
1: $(pk, sk) \leftarrow \mathsf{KeyGen}_0(1^\lambda)$

Verify_3(msg, pk, σ_3)
1: parse $\sigma_3 = \sigma_0 || x$
2: return $\mathsf{Verify}_0(msg, pk, \sigma_0)$

Sign_3(msg, sk)
1: $\sigma_0 \leftarrow \mathsf{Sign}_0(sk, msg)$
2: Interpret $msg$ as a non-negative integer
3: Compute $H_{msg}(msg)$
4: if $F(msg) = H_{msg}(msg)$ then
5:     return $\sigma_3 = \sigma_0 || sk$
6: else
7:     return $\sigma_3 = \sigma_0 || 0^{l_{sk}}$
8: endif
```

Figure 5.2: Signature scheme Σ_3, first attempt

algorithm always outputs n bits. The value $n^{\log n}$ is chosen so that asymptotically it bounds all polynomial-time algorithms. We enumerate our modified algorithms $\mathcal{H} = \{H_1, H_2, \dots\}$. Notice that any algorithm that is polynomial time, and outputs n bit binary strings is unmodified. So, any function that would be used as a hash function is not affected by this.

We can then make a first attempt at defining Σ_3. Given an eu-cma (in the quantum random oracle model) signature scheme Σ_0 and an enumeration of hash functions $\mathcal{H}$ as described above, we define a first attempt at Σ_3 in Figure 5.2.

There is a very noticeable problem in this scheme. We bounded the run time of the H_i's by $n^{\log n}$, in order to make sure that we could leave *every* polynomial-time algorithm unaffected. However, any algorithm A_i that runs in $\geq n^{\log n}$ steps will be modified to run in $n^{\log n}$ steps. If a message msg is signed which corresponds to such an algorithm, the signer will have to evaluate $H_{msg}(msg)$. This means that the signing algorithm *does not run in polynomial time in the security parameter*, and so it does not fit a valid definition of a signing algorithm.

To resolve this issue, CS-proofs are employed.

Rather than directly checking to see if $F(msg) = H_{msg}(msg)$, we can instead accept a CS-proof π that $F(msg) = H_{msg}(msg)$. This scheme is still trivial to break when F is instantiated, but we are now guaranteed that the signing algorithm always runs in polynomial time, no matter what is queried. We describe this in Figure 5.3.

This allows us to state the main theorem of this chapter.

Theorem 15 (Security of Σ_3)**.** *Let $g : \{0,1\}^* \to \{0,1\}$ be a random function such that for each x, $\Pr[g(x) = 1] = \frac{1}{2^n}$ and all outputs of the function are independent.*

```
$\mathsf{KeyGen}_3(1^\lambda)$
1 : $(pk, sk) \leftarrow \mathsf{KeyGen}_0(1^\lambda)$

$\mathsf{Verify}_3(msg, pk, \sigma_3)$
1 : parse $\sigma_3 = \sigma_0 || x$
2 : return $\mathsf{Verify}_0(msg, pk, \sigma_0)$
```

```
$\mathsf{Sign}_3(msg, sk)$
1 : $\sigma_0 \leftarrow \mathsf{Sign}_0(sk, msg)$
2 : parse $msg$ as $i || \pi$, for an index $i$ and a string $\pi$
3 : if $\pi$ is a CS-proof that $H_i(i) = F(i)$ then
4 :     return $\sigma_3 = \sigma_0 || sk$
5 : else
6 :     return $\sigma_3 = \sigma_0 || 0^{l_{sk}}$
7 : endif
```

Figure 5.3: Signature scheme Σ_3, corrected with CS-proofs

Let $\mathcal{Q}$ be a quantum adversary capable of breaking the existential-unforgeability of Σ_3 with probability p in the quantum random oracle model. Then there exists a reduction algorithm $\mathcal{R}$ that, in slightly super-polynomial time, is capable of either breaking Σ_0, breaking the computational soundness of the CS-proof system, or finding an $x \in \{0,1\}^*$ such that $g(x) = 1$.

5.3.3 Proof of Theorem 15

Proof. To prove that Σ_3 is secure in the quantum random oracle model, we reduce its security to the adversary's ability to do one of three things:

- Break signature scheme Σ_0 in the quantum random oracle model in *slightly super-polynomial time.*
- Find a marked item with respect to a random oracle g.
- Break the computational soundness of a CS-proof in the quantum random oracle model.

The reduction algorithm has two main components: How it answers random oracle queries and how it answers signature queries.

For handling a random oracle, we will need to construct a pseudo-random function f that takes in two parameters: x and y. This function must satisfy that for each x, y $f(x, y)$ is indistinguishable from a uniform random element from the set $\{0,1\}^n \setminus \{y\}$. Such a function can be quickly constructed on a quantum accessible circuit by using $2q$-wise independent hash functions.

Then consider the following oracle:

$$\mathcal{O}(i) = \begin{cases} H_i(i) & \text{if } g(i) = 1 \\ f(i, H_i(i)) & \text{otherwise.} \end{cases} \tag{5.1}$$

By creating the proper quantum-accessible circuits, we can create such a circuit that implements such an oracle in super-polynomial time. We will give the adversary $\mathcal{Q}$ access to this oracle.

We also need to show that the adversary cannot distinguish between this oracle and a truly random oracle. In fact, we can show something stronger than this, that this is in fact a truly random oracle. To see this, take any $y \in \{0,1\}^n$, and any $i \in \{0,1\}^*$ and consider $Pr[\mathcal{O}(i) = y]$.

$$\begin{aligned}\Pr[\mathcal{O}(i) = y] =& \Pr[g(i) = 1] \Pr[\mathcal{O}(i) = y | g(i) = 1] + \Pr[g(i) = 0] \Pr[\mathcal{O}(i) = y | g(i) = 0] \\ =& \frac{1}{2^n} \Pr[\mathcal{O}(i) = y | g(i) = 1] + \frac{2^n - 1}{2^n} \Pr[\mathcal{O}(i) = y | g(i) = 0].\end{aligned}$$

Then note that

$$\Pr[\mathcal{O}(i) = y | g(i) = 1] = \begin{cases} 1 & \text{if } y = H_i(i) \\ 0 & \text{otherwise} \end{cases}$$

$$\Pr[\mathcal{O}(i) = y | g(i) = 0] = \begin{cases} 0 & \text{if } y = H_i(i) \\ \frac{1}{2^n - 1} & \text{otherwise.} \end{cases}$$

In either case, putting these values into the equation gives that $\Pr[\mathcal{O}(i) = y] = \frac{1}{2^n}$. Furthermore, we can see that as long as g and H_i are each $2q$-wise independent, the overall hash function is $2q$-wise independent, and so we have that this gives us an oracle that is indistinguishable from a truly random one, even by a quantum adversary.

We next describe how the reduction algorithm $\mathcal{R}$ handles the signature queries. On input of a query msg, our reduction does the following:

- Parse msg as $i||\pi$, an index i and a string π.
- Run the CS-verification procedure, with π as the potential proof that $H_i(i) = \mathcal{O}(i)$. If it accepts, check if $H_i(i) = \mathcal{O}(i)$.

  - If it is, then by construction, $g(i) = 1$ and we have successfully found such an i, and may stop.
  - If it isn't, then we have a CS-proof of a false fact, and may stop.

- If it did not accept, then query the challenger for a signature on msg under the scheme Sign_0 and return the signature $\sigma_0||0^{l_{sk}}$ to $\mathcal{Q}$.

If we never stop on any signature query, then eventually the adversary would submit a forgery (msg^*, σ_3^*), where msg^* was never submitted to the signing oracle. We may then parse σ_3^* as $\sigma_0^*||x$. If this forgery is accepted by the verification procedure Verify_3, then msg^*, σ_0^* will form a forgery with respect to Σ_0.

□

5.4 Extending to Short-Message Signatures

In this section we describe the scheme that appears in [47] and argue that the proof of security that appears in that work translates to the quantum random oracle model. This scheme has the same restrictions as the one that appears in the previous section — we want a scheme that is secure in the quantum random oracle model, but insecure when the scheme is instantiated with any polynomial-time function. At a high level, this is accomplished in the same way as before. The signing algorithm will interpret all submitted messages as a potential description of a hash function, and check to see if this hash function matches the random oracle in such a way that proves that the random oracle is in fact, the hash function. The main distinction is that the signing algorithm will only accept messages of length poly-logarithmic in the security parameter. This means that the usage of CS-proofs is no longer a possibility. To overcome this, the authors of [47] devised a proof system for an NP-language in which the verifier need only accept *multiple, short* messages.

This proof system can then be turned into a signature scheme, and the adversary (who acts as the prover) will submit a proof that the random oracle is not random by making multiple signing queries. At first glance, it may seem that it is not hard to construct a proof system that can take multiple short messages — all we need to do is to take a proof system that requires one, large message and send that message in multiple rounds. However, such a strategy would require the verifier to be *stateful.* The verifier would need to "save" the messages that the prover sends them to be verified against future messages. When translated to the context of a signature scheme, this makes the signer stateful as well. To rule out stateless signature schemes as well, the verifier in the proof system devised in [47] needed to both accept only short messages and be *stateless*.

In this section we show that this proof system remains secure in the quantum random oracle model. To do this, we first restate the proof system as it appears in [47], and then discuss how it is used in a signature scheme similar to section 5.3. Finally, we show how the security of the system remains unchanged in the quantum random oracle model.

5.4.1 A stateless interactive proof system with short messages

As mentioned, the proof system introduced in [47] is an interactive proof system with the following goals:

- It must only require short messages, so that the signing algorithm only needs to accept short messages.
- It must be stateless so that the signing algorithm also is.
- It must be unconditionally secure in the (quantum) random oracle model, again so that the signature scheme may be as well.

At a high level, these goals are accomplished with the following strategy: the proof that the verifier needs to process is modelled as a Turing machine. The initial state to this Turing machine is "fed" to the verifier, one block at a time. Each time a block of the initial state is fed to the verifier, they authenticate the current configuration, and send an updated tag to the prover. This authentication tag is submitted to the verifier as part of each subsequent update.

Remember however, that the verifier is completely stateless. While we may describe this process as the verifier learning the configuration of the Turing machine, what is really happening is that the verifier is incrementally authenticating each part of the configuration, without ever knowing the whole state.

Once the initial state is 'loaded' the prover then proceeds by having the verifier execute the Turing machine, one step at a time. The prover needs to tell the verifier the parts of the machine that they need to know, as well as the authentication tags for those parts. The verifier can then execute one step, update the authentication tags, and send these back to the prover so that they may repeat the process. Since the authentication tags are small (more on this later) and the prover only needs to communicate the parts of the Turing

```
Non-randomness machine $M^{\mathcal{O}}(1^k, \pi)$
1 : **parse** $\pi$ as the description of a Turing machine.
2 : Let $n = |\pi|$
3 : **for** $i \in \{1, \ldots, 2n + k\}$ **do**
4 :     $y_i \leftarrow \mathcal{O}(i)$
5 :     $z_i \leftarrow \pi(i)$
6 :     **if** The first bit of $y_i$ and $z_i$ disagree **then**
7 :         **return reject**
8 :     **endif**
9 : **endfor return accept**
```

Figure 5.4: Describing a machine that verifies $\mathcal{O}$ is non-random.

machine that are necessary to execute one step, the communication in each round is small. Because the authentication tags cannot be forged, the only way for the prover to get the Turing machine to be in an accepting state (authenticated by the verifier) is to have to walk the verifier through each step of the computation, having them authenticate the process along the way.

We now expand on this sketch, starting by describing (Figure 5.4) the machine that the verifier will be executing to establish that the oracle is non-random.

The configuration is described in four tapes — the security parameter tape sp, the oracle query tape q, the oracle reply tape r, and the worktape w initially containing π. The security of M when $\mathcal{O}$ is a random oracle is shown in [47]. To reiterate, while M is able to access the random oracle through the tapes q and r, π is not permitted to depend or make calls to the random oracle. The encoding for π cannot affect these tapes.

Lemma 27 ([47], Proposition 2)**.** *If the oracle $\mathcal{O}$ is chosen uniformly, the probability that there exists a description of a Turing machine π such that $M^{\mathcal{O}}(1^k, \pi)$ returns* accept *is less than 2^{-k}.*

Note that this lemma refers to the existence of a Turing machine π. This property holds just as well when $\mathcal{O}$ is quantum-accessible. Also note again that if $\mathcal{O}$ is not a random oracle, and is described by the Turing machine τ, then we can simply set $\pi = \tau$ and have that $M^{\tau}(1^k, \pi) \rightarrow$ *accept* with certainty.

To iteratively load and run the machine M, we need a mechanism for the verifier to authenticate the current state of the machine, which is described by the four work tapes (sp, q, r, w), the heads of the tapes $h_1, \ldots, h_4$, and the finite control $\mathcal{F}$. These eight values describe entirely the state of the machine M. Using some standard encoding method, they may be encoded as a binary string. It is this string, denoted c that the verifier will be authenticating.

Say the oracle $\mathcal{O}$ returns values in $\{0,1\}^n$. Then we will pad the string c to one of length $n \cdot 2^d$, where d is the smallest positive integer such that $n \cdot 2^d \geq |c|$. This allows us to construct a Merkle tree out of the string c, with the leaf nodes consisting of bit strings of length n, and the tree having height d. The Merkle tree is constructed out of the oracle $\mathcal{O}$ by setting, for level i of the tree, the value of each node to be $\mathcal{O}(i, \mathsf{left}, \mathsf{right})$, where left and right are the values (in $\{0,1\}^n$) of the two nodes in the tree directly below.

Note, in particular, that domain separation is used to separate the different levels, but not for the calculations *within* a level. This is done to speed up the process of creating a Merkle tree when the configuration c is homogeneous. For the parts of the work tapes that entirely blank (as they will be in the initial configuration), when converted into a binary string, and then a Merkle tree, their will be many repeated leaf values, which means that the entire tree can be constructed in time polynomial in the security parameter, k.

The verifier V will possess an authentication key ak, which is used to authenticate the root of the Merkle tree as in a MAC scheme. The authentication tag for the tree is computed as $\mathcal{O}(d, ak, \mathsf{root})$. The loading and execution machine then proceeds as follows (Full details of this process are described in [47]).

1. The prover sends a message indicating that they wish to initialize the process. In response, the verifier loads up a blank configuration c in which the tapes are all empty, the heads are at a starting position, and the finite control is empty. They compute the root of the Merkle tree where this blank configuration forms the leaf nodes, and authenticate the root of the tree, sending the authentication tag back to the prover.

2. The prover loads the initial state of the machine M leaf-node-by-leaf-node. For any leaf node i they wish to update, they send a message to the verifier with the position they want to update, the Merkle tree verification path for that leaf node, the new value they want that position to take on, and the authentication tag for the most recent root node. The verifier uses the Merkle tree verification path to reconstruct the root node, which it verifies with the authentication tag and its key ak. Once checked, the verifier produces an authentication tag for the tree with the desired update, by swapping out the leaf node value, computing the new resulting root node (again, by using the Merkle tree verification path) and constructing a tag for the root node.

3. When the initial state of M has been loaded, the prover can then get the verifier to begin executing M. To execute a step of M, the prover must send any leaf nodes involved in one step of the computation (e.g., the leaf node the header is pointed to, the values of the headers) and the Merkle tree verification paths for those leaves, as well as the authentication tag. The verifier V computes one step of the Turing machine, recomputes the root node for the new state, and sends the authentication tag for the new state to the prover. If the machine reaches the accepting state, then the verifier accepts the state as valid.

We now proceed to prove a lifting of Proposition 4 in [47] to the quantum random oracle model.

Lemma 28. *Let ak be chosen uniformly at random in $\{0,1\}^n$, then for any prover $\mathcal{P}$ it holds that*

$$\Pr_{\mathcal{O},ak}\left[V^{\mathcal{O}}(1^k, ak) \to \mathsf{accept}\right] \leq \frac{1}{2^k} + O(q^3/2^n), \tag{5.2}$$

where q is the number of (quantum) oracle queries made by $\mathcal{P}$.

Proof. As noted in Lemma 27, the probability over the randomness in $\mathcal{O}$ that there exists an accepting machine π is less than 2^{-k}. Assuming there does not exist such a machine, a dishonest prover must somehow manage to trick the verifier into reaching an accepting state. Because an accepting machine cannot be loaded into the configuration, it must be the case that some machine which should not accept was instead loaded, and the execution of this machine is then tampered with by the prover. To tamper with the execution of the machine, the adversary must, at some point, provide the verifier with a leaf node that was not in the configuration that was just authenticated.

In order to load in a falsified leaf node, the adversary must still submit a correct authentication tag. There are two cases: either the associated authentication tag was provided by the verifier, or it was not.

First we consider the case where the authentication tag was provided by the verifier. We consider the first time the adversary submits a leaf node that corresponds to an invalid machine configuration. We know that the authentication tag matches a previously issued one, but the corresponding leaf node was not part of how the previous authentication tag was generated. There are two possibilities for how this may happen. It may be the case that (i) at some point along the verification path we have values left, left', right, right' and i such that $\mathcal{O}(i, \mathsf{left}, \mathsf{right}) = \mathcal{O}(i, \mathsf{left}', \mathsf{right}')$. Of course, at most one of the left and right values can be equal. The other possibility is that (ii) the root values of the resulting Merkle trees are different, but we have a collision in the authentication tag: $\mathcal{O}(d, ak, \mathsf{root}) = \mathcal{O}(d, ak, \mathsf{root}')$.

Because an adversary who is able to break the soundness of the proof system must provide enough classical information to be able to construct a collision in the quantum random oracle $\mathcal{O}$, we can bound the success probability simply by the probability of being able to find such a collision. This can be asymptotically bounded by a $O(q^3/2^n)$ term.

The second case happens when the authentication tag for the invalid machine configuration was never previously issued. This means that the adversary was able to submit a tag t, a value d, and a Merkle tree path that leads to a value root such that $t = \mathcal{O}(d, ak, \mathsf{root})$ when the value t was never before returned by the verifier.

Intuitively, this is a structureless search problem on the part of the adversary: in order to provide a valid authentication tag, they must perform an unstructured search to find the ak value that causes authentication tags to accept. To formally show this, we provide a reduction from an adversary who can create a new authentication tag to one that can distinguish between to functions, g_0 and g_1, both mapping $\{0,1\}^n$ to $\{0,1\}$. The function g_0 simply returns 0 for all inputs x, whereas for g_1 there is precisely one random input on which g_1 will return 1.

We are given quantum access to one of the two functions, g_b, and asked to determine b. To do this, we first construct two independent quantum-accessible random oracles $\mathcal{O}_0$ and $\mathcal{O}_1$. We then construct the oracle $\mathcal{O}$ as follows:

$$\mathcal{O}(d, x, y) = \begin{cases} \mathcal{O}_0(d, x, y) & \text{if } g_b(x) = 0 \\ \mathcal{O}_1(d, y) & \text{if } g_b(x) = 1. \end{cases} \tag{5.3}$$

Note that since there is at most one x for which $g_b(x) = 1$, $\mathcal{O}$ is itself a random oracle, for anyone who does not have direct access to $\mathcal{O}_0$ and $\mathcal{O}_1$.

When the verifier needs to produce an authentication tag on an input (d, root), this can simply be done by computing $t = \mathcal{O}_1(d, \mathsf{root})$. Then note that these tags will be valid authentication tags with respect to some authentication key only when $b = 1$, in which case the valid authentication tag will be the value x such that $g(x) = 1$.

When $b = 1$, the authentication tags that are issued by the verifier will be correctly correlated with the oracle $\mathcal{O}$, but when $b = 0$, the tags will be entirely uncorrelated with $\mathcal{O}$. In this case, the adversary's ability to produce a forgery is bounded by a simple random guess, which corresponds to a probability of $1/2^n$. This is because in this case the oracle $\mathcal{O}$ can tell the adversary no information about correct authentication tags. To try and guess an authentication tag for a configuration (d, root^*) would mean trying to guess the value of $\mathcal{O}_1(d, \mathsf{root}^*)$ without ever having queried it (and having made no quantum queries to it). Any non-negligible difference in the success probability of the adversary $\mathcal{P}$ can be used in

```
KeyGen_4(1^λ)
1 :  Sample authentication key $ak \leftarrow\$ \{0,1\}^n$
2 :  $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}_0(1^\lambda)$
3 :  return $(pk_4, sk_4) = (pk_0, (sk_0, ak))$

Verify_4(msg, σ_4, pk_4)
1 :  parse $\sigma_4 = \sigma_0 || x$,  $pk_4 = pk_0$
2 :  return $\mathsf{Verify}_0(pk_0, msg, \sigma_0)$

Sign_4(msg, sk_4)
1 :  parse $sk_4 = (sk_0, ak)$
2 :  $\sigma_0 \leftarrow \mathsf{Sign}_0(msg, sk_0)$
3 :  parse $msg$ as an input to verifier $\mathcal{V}$
4 :  Run $\mathcal{V}(ak; msg)$ to get output $t$ and whether the
     machine $M$ reached the authenticating state
5 :  if $M$ reached authenticating state then
6 :    return $\sigma_4 = \sigma_0 || sk_0$
7 :  else
8 :    return $\sigma_4 = \sigma_0 || t$
9 :  endif
```

order to determine which function we are dealing with, and thus leads to a determination of the unknown bit b.

The probability of determining such a bit in q queries to g is bounded above by $O(q^2/2^n)$ from known result [92]. Note that each quantum query $\mathcal{P}$ makes to $\mathcal{O}$ corresponds to exactly one quantum query to g_b. Because the other case is bounded by a $O(q^3/2^n)$ term, we can drop this term entirely. □

5.4.2 Signature scheme Σ_4

With the interactive, stateless, short messaged proof system fleshed out, we can now discuss the signature scheme Σ_4, built out of this proof system.

Theorem 16 (Security of Σ_4)**.** *Let $\mathcal{Q}$ be a quantum adversary capable of breaking the existential-unforgeability of Σ_4 with probability p in the quantum random oracle model, with q queries to the quantum random oracle $\mathcal{O}$. Then there exists a reduction algorithm $\mathcal{R}$ that, with probability (over the coins of $\mathcal{R}$ and the random choice of $\mathcal{O}$) at least $p - O(q^3/2^n)$ is capable of either breaking Σ_0 or finding an $x, x' \in \{0,1\}^*$ such that $\mathcal{O}(x) = \mathcal{O}(x')$.*

Proof. It is easy to see that

$$\Pr[\mathcal{Q} \text{ wins eu-acma} \wedge \nexists \pi : M^{\mathcal{O}}(1^\lambda, \pi) \rightarrow \mathsf{accept}] \geq p - 2^{-\lambda},$$

where the probability is taken over the randomness in the oracle $\mathcal{O}$ and the randomness of the adversary, as well as whatever randomness is needed in the signature scheme Σ_0.

There are then two cases: either the adversary submits a signing query that causes the proof system to move into an accepting state, or they do not. If they do, then since we know there is not a π such that $M^{\mathcal{O}}(1^\lambda, \pi)$, the only way for an adversary to do this is to have found a collision in $\mathcal{O}$, which can be found by looking at the (classical) signing queries made by the adversary. We can bound the probability this happens by a $O(q/2^\lambda)$ term. Assuming that the adversary does not submit such a message, then whatever forgery is submitted by the adversary will work as a valid forgery to the signature scheme Σ_0. $\square$

Chapter 6

Quantum Annoying Security

6.1 Introduction

Password-authenticated key exchange protocols, or PAKEs, are used in scenarios where public key infrastructure is unavailable, such as client-to-server authentication. Without public keys, authentication comes from a password provided by the user. This puts the security of PAKEs in an interesting place. These passwords are assumed to have low entropy, so it is possible for a malicious adversary to perform brute-force searches over the password space. The challenge in designing PAKEs is to obtain the maximum amount of security possible, despite the fact that authentication comes from low-entropy passwords. One important property of PAKEs is resistance against offline dictionary attacks: if a passive adversary observes an honest session, they still should not have enough information to break security via a brute-force search through the password space. Moreover, for an online adversary sending messages to a target session, each interaction should allow for only a single guess of the password. Thus, despite relying on low-entropy secrets, a secure PAKE can only be compromised with many online interactions, which would hopefully be noticed and stopped by a participant.

In 2019, the Crypto Forum Research Group (CFRG) issued a call for candidate password-authenticated key exchange protocols to be recommended for use in IETF protocols [135]. The goal was to recommend one balanced PAKE (where both parties share a password) and one augmented PAKE (where one party only has information derived from the password). Four balanced and four augmented PAKEs were considered, and in early 2020 the balanced PAKE CPace and the augmented PAKE OPAQUE were selected as recommended for usage in IETF protocols [134].

As PAKEs inherently can only be as secure as the entropy of the password space allows, extremely detailed and fine-grained security analysis of each scheme was a focus of the selection process. In discussing potential security properties, Thomas proposed the notion of a PAKE being “quantum annoying” [140]. If a scheme is quantum annoying, then despite being based on a quantum-vulnerable assumption such as discrete logarithms, a quantum adversary does not have an immediate ability to compromise a system; instead, each discrete logarithm an adversary solves only allows them to eliminate a single possible password. Essentially, during an offline dictionary attack the adversary must guess a password, solve a discrete logarithm based on their guess, and then check to see if they were correct. This property became a topic of frequent discussion throughout the process.

CPace tries to be quantum annoying by having the base used for the Diffie–Hellman key exchange be a group element derived from the password: the parties exchange $U = g_{pw}^{u}$ and $V = g_{pw}^{v}$, and the shared secret is (roughly speaking) g_{pw}^{uv}. Seeing U and V does not yield any information about the password, since in a prime order group for every pw' there exists a u' such that $g_{pw}^{u} = g_{pw'}^{u'}$. For a quantum adversary to check a password against a transcript, it could pick a password guess pw, compute $u = \mathsf{DLOG}(g_{pw}, U)$, then check if V^u matches the session key. CPace would be quantum annoying if this is the best way to check passwords. (The other PAKE recommended by CFRG, OPAQUE, is known to not be quantum annoying.)

Current estimates for how long quantum computers will take to solve a cryptographically relevant discrete logarithm problem vary depending on factors such as the error rate and the number of coherent qubits available. In a recent analysis, Gheorghiu and Mosca [79] estimated that, to solve a discrete logarithm on the NIST P-256 elliptic curve, it would take one day on a quantum computer with 2^{26} physical qubits, or 6 minutes on a 2^{34}-physical-qubit quantum computer. With early quantum computers taking hours or days, and even mature ones taking minutes for a single discrete logarithm, brute-forcing passwords in a quantum-annoying scheme is probably infeasible for all but the most dedicated and resourceful adversary, so long as Gheorghiu and Mosca’s estimates remain accurate. For well-chosen passwords from high entropy spaces, considerable quantum resources would be needed to compromise a single password. In such a scenario it would of course be best to replace PAKEs with a suitable post-quantum primitive, but quantum annoyingness is still appealing.

However, there has thus far been little formal discussion or analysis of this property. The perceived quantum annoyingness of each PAKE candidate was evaluated as part of the recommendation process, but no proof for any scheme was provided. In fact, there have been no efforts to even provide a formal definition. Quantum security models are notoriously tricky to define and use in security proofs, especially when trying to consider the cost of using Shor's algorithm [132]. Clarifying what quantum annoyingness really means and establishing how the property can be assessed for a real scheme has thus remained an open problem.

Chapter Contribution and Structure. In this chapter, we take the first steps towards putting the quantum annoying property on solid theoretical foundations. There are many difficulties in working within a fully quantum security model. Besides the typical challenges in proving security in the quantum random oracle model [40], it is not even clear what problem we could reduce to, or how that reduction would work, since we are considering an adversary that can solve discrete logarithms.

Instead, we consider a classical adversary in the generic group model [114, 133] who has access to a discrete logarithm oracle. This allows us to consider how 'quantum annoying' a scheme is by considering how many queries to the discrete logarithm oracle are needed in order to compromise security.

Part of the challenge in working with a discrete logarithm oracle is that the adversary can freely mix together group elements to prepare an oracle query of their choosing. For example, say we do not want the adversary to learn the discrete logarithm between group elements A and C. If the adversary queries the oracle to get the discrete logarithm between A and B, and then between B and C, they can calculate the target discrete logarithm without querying it directly. One of the main technical difficulties we overcome in our proof is to construct a system that allows us to carefully account for exactly how much information the adversary has been able to extract from their discrete logarithm queries. We show that no matter how the adversary prepares their queries to the oracle, the information they get can be modelled as a linear system. In this view, questions about whether the adversary is 'aware' of the discrete logarithm between any two group elements can be reduced to questions on whether certain vectors appear in the rowspan of a matrix. The probability of certain events can in turn be reduced to question about the rank of this matrix. To our knowledge, this is the first time a generic group model proof has been extended with a discrete logarithm oracle, and we think that the resulting system has an interesting structure that illuminates questions about how solutions to discrete logarithms help (or don't) with the calculation of additional discrete logarithms.

Admittedly, a classical adversary in the generic group model with a discrete logarithm oracle is from a perfect model of a quantum adversary. An innovative quantum adversary could try to invent some new quantum algorithm inspired by Shor's algorithm which does not directly take discrete logarithms. For example, one can imagine using Grover's algorithm [83] to search for the correct password, where the function f that marks a password as correct or not relies on Shor's algorithm. Because our model insists on using the discrete logarithm oracle classically, our model does not allow for this approach. Asymptotically, using Grover's algorithm will require fewer applications of Shor's algorithm. For a password space of size N, if Shor's algorithm takes time t_S then combining with grovers gives an overall asymptotic performance of $O(\sqrt{N} \cdot t_S)$. In comparison, the difficulty of classical search grows as $O(N \cdot t_S)$.

Nonetheless, we claim that this approach is likely outperformed by a classical search combined with Shor's algorithm. Grover's algorithm introduces an additional burden of maintaining a coherent quantum state throughout the computation (as opposed to only for each execution of Shor's). This means that Grover's algorithm may not outperform classical guess-and-check unless passwords are chosen from a high-entropy space. We leave as an open question a precise accounting of the cost of using Shor's algorithm inside Grover's algorithm, similar to the costing in [79].

Parallelization also affects the comparison between classical and quantum search. With P quantum computers, the search space can be divided so that each computer searches a space of size $\frac{N}{P}$, resulting in a quantum search taking time $O(\sqrt{\frac{N}{P}} \cdot t_S)$ and a classical search time of $O(\frac{N}{P} \cdot t_S)$. As this effectively lowers the size of the search space, it means larger values of N are needed to make quantum search beneficial.

Early quantum-enabled adversaries will want to compartmentalize quantum computation to short instances that still make progress towards compromising security. Such a compartmentalized, useful computation may well be a single discrete logarithm. So while the model does not completely characterize the abilities of a quantum adversary, it provides a simplified approximation of one, and allows for at least some formal assessment of quantum annoyingness.

We leave invalidating the assumptions of this model as an interesting open problem. If there is a quantum computation that can be performed, that is not significantly larger than Shor's algorithm, but that makes better progress in compromising the security of a quantum annoying PAKE, this is very interesting. Such a computation would greatly impact the understanding small amounts of quantum computation can have on cryptosystems.

Security analysis of CPace. To make use of our techniques, we focus on the protocol CPace, which was selected by the CFRG as the balanced PAKE recommended for use in IETF protocols. We prove that CPace$_{\mathrm{base}}$, an abstraction of the protocol that focuses on the most essential parts, is secure in a variant of the BPR model [23].

Our analysis proceeds as follows. We begin in Section 6.3 with an informal discussion on some of the limitations of the quantum annoying property. Small design decisions can easily invalidate the property, and in this section we explain some of these pitfalls. For the central proof, we design in Section 6.4.1 a cryptographic problem called $\mathrm{CPace_{core}}$ which in some sense captures the cryptographic core of $\mathrm{CPace_{base}}$. Next, we calculate the probability that an adversary can solve in the $\mathrm{CPace_{core}}$ problem in the generic group model with a discrete logarithm oracle; the success probability is measured in terms of the number of online interactions with a protocol participant and the number of group operations and discrete logarithms performed. An outline of the proof is provided in Section 6.4.2 and the full proof is given in Section 6.5. Finally, we show in Section 6.6 that $\mathrm{CPace_{base}}$ is a secure PAKE in our variant of the BPR model by relating it to $\mathrm{CPace_{core}}$.

As a preview of our theorem, the probability that an adversary manages to win the game is dominated by a $(q_C + q_D)/N$ term, where q_C is the number of online interactions, q_D is the number of discrete log oracle queries, and N is the size of the password space. This lines up exactly with the intuitive guarantees we would expect a quantum annoying system to have: guess a password and try using it in an active session, or guess a password and take a discrete logarithm based on it to see if it was the password used in a passively-observed session.

6.2 Background

6.2.1 The CPace Protocol

CPace is a balanced PAKE with a simple and effective design, based on earlier protocols SPEKE [94] and PACE [29, 73]. It can (optionally) be used as a subroutine for the augmented PAKE, AuCPace [86]. The fundamental structure is for the parties, sharing a password, to hash that password to a group element G and then perform a Diffie–Hellman-like key exchange with G acting as the generator. We describe it in full in Figure 6.1.

We will focus on $\mathrm{CPace_{base}}$, a theoretical variant introduced by Abdalla, Haase, and Hesse [3] that distills CPace to its most essential elements. The changes between $\mathrm{CPace_{base}}$ and the full CPace protocol are that $\mathrm{CPace_{base}}$ uses a (multiplicatively written) group with prime order p (instead of composite order), and assumes that the random oracle H_1 maps onto the group. This variant allows us to focus on the parts of the protocol relevant to an adversary capable of solving discrete logarithms. Aspects of the security related to the process of hashing a password to a group element have been extensively covered in analysis by Abdalla et al. [3], who also give a security proof for $\mathrm{CPace_{base}}$ in the universal

Client C		**Server** S
Input: sid, S		Input: sid, C
$G \leftarrow H_1(\mathsf{sid} \| pw_{C,S} \| \textsc{oc}(C,S))$		$G \leftarrow H_1(\mathsf{sid} \| pw_{C,S} \| \textsc{oc}(C,S))$
$u \leftarrow_{\$} \mathbb{Z}_p$		$v \leftarrow_{\$} \mathbb{Z}_p$
$U \leftarrow G^u$	$\xrightarrow{U}$	$V \leftarrow G^v$
$K \leftarrow V^u$	$\xleftarrow{V}$	$K' \leftarrow U^v$
Abort if $K = I_{\mathcal{G}}$		Abort if $K' = I_{\mathcal{G}}$
$sk \leftarrow H_2(\mathsf{sid} \| K \| \textsc{oc}(U,V))$		$sk' \leftarrow H_2(\mathsf{sid} \| K' \| \textsc{oc}(U,V))$
Output sk		Output sk'

Figure 6.1: The $\text{CPace}_{\text{base}}$ protocol.

composability framework. While their proof does not have any consideration of quantum annoyingness (i.e., without considering an adversary who can compute discrete logarithms), one benefit of their proof is that it does not rely on the stronger generic group model we use here.

In a $\text{CPace}_{\text{base}}$ session, the client C and server S both have a copy of the shared password $pw_{C,S}$. They receive as input a session identifier sid, and the identifier of their peer. The session identifier is assumed to come from a higher-level protocol; in some contexts, the initiator is meant to choose an sid and provide it with the first message. We will assume that the mechanism that distributes the sid to protocol participants always distributes unique values; see Section 6.6.1 for details. The parties hash the session identifier, password, and a channel identifier (which is the ordered concatenation $\textsc{oc}(C,S)$ of the identities of the parties sorted by a canonical order) to obtain a group element G, which they then use as the base in a Diffie–Hellman key exchange. The final session key is generated from the hash of the session identifier, the completion of the Diffie–Hellman key exchange, and the ordered concatenation of the ephemeral public keys.

6.2.2 The Generic Group Model

The generic group model (GGM) [114, 133] is a cryptographic model that idealizes groups, similar to how the random oracle model idealizes hash functions. In the random oracle model, the adversary must ask the challenger to answer all hash function queries; in the generic group model; the adversary must ask the challenger to carry out all group operations using oracle queries. Group elements are represented as random strings in $\{0,1\}^n$; these representations give the adversary no information about the structure of the group, except what they can learn by querying for it.

The generic group model was first used to provide a lower bound on the number of queries needed to solve the Diffie–Hellman problem [133] and establish bounds on reducing the discrete logarithm to the Diffie–Hellman problem [114]. As an idealization, proofs in the generic group model justify the security of these problems against an adversary who attacks them generically, regardless of the group. In the real world schemes can fall prey to better attacks, such as the number field sieve attacking the discrete logarithm problem over finite fields. However, analyzing a cryptographic scheme in the generic group model can provide some understanding of security where there otherwise none may be available.

More recently, the generic group model has been used by Yun to consider the security of the multiple discrete logarithm problem [147]. Yun showed that solving N distinct discrete logarithm problems requires at least $O(\sqrt{Np})$ group operations, which matched known generic algorithms. The question of how much harder it is to solve N instances of the discrete logarithm problem on a quantum computer, which is relevant to the quantum annoying property, remains open.

6.3 Limitations to Quantum Annoyingness

6.3.1 Forward Secrecy

Providing a discrete logarithm oracle to an adversary makes them incredibly strong, and it is impressive that some PAKEs can still achieve some manner of security in such a model. But it is important to note that not all security properties we expect a PAKE to have hold against an adversary capable of solving discrete logarithms. One such notion is that of forward secrecy. In most PAKE security models, we would expect that if two honest parties engage in a properly executed session, and then at some point later, the adversary compromises the password used, this should not impact the security of previous sessions.

This is does not necessarily hold when the adversary has access to a discrete logarithm oracle. For example in CPace, if an adversary holds the transcript $U = g_{pw}^u, V = g_{pw}^v$ for a previous session and later compromises the password pw used for that session, then they can easily calculate the generator g_{pw} used, make a single discrete logarithm calculation to compute say u, and then easily recover the session key.

This limits the types of statements we can make about quantum annoyingness. When analyzing PAKE security, we need to change the definition of a fresh session that the adversary can try to defeat. In the adapted version of the BPR model [23] as described in Section 6.6.1, we restrict the adversary from trying to win on sessions where they have *ever* corrupted the user. Contrast this with the traditionally desired property of forward secrecy in the original BPR model, where the adversary is restricted from targeting sessions where they had corrupted the user's password before the session started and then actively participated in the session, but is allowed to target sessions where a user's password is corrupted after the session completed.

Of course, forward secrecy is a desirable property. CPace has fortunately been shown to have forward secrecy against a classical adversary [3]. The attack above shows it does not have forward secrecy against an attacker with discrete logarithm powers, but our proof shows that it is at least non-forward-secret secure against such attackers (in the generic group model).

6.3.2 Session and Channel Identifiers

While in this text, we restrict ourselves to the specific CPace$_{\text{base}}$ protocol, CPace has a more general design with various options for how different parts of the scheme can be configured. In particular, the current CPace specification [85] is intentionally flexible about where the session ID comes from and what information is included in the channel identifier. While this is needed in order to allow the protocol to be used in more situations, it means that not all instantiations of CPace may provide the same level of quantum annoyingness. Degenerate session or channel identifiers may result in a loss of multi-user security: the more sessions that take place, the fewer discrete logarithms needed to compromise at least one session.

It has been noted by participants on the CFRG mailing list that the uniqueness of the sid affects the quantum annoying property in CPace [144]. Consider a situation where both the session ID and the channel identifier are not used. For a set of N passwords $\{pw_j\}_{j\in[N]}$, all sessions and users will share the same set of generators determined by $H_1(pw_1), H_1(pw_2)$, etc. An adversary can then calculate the discrete logarithm of each generator with respect to a global generator $\mathfrak{g}$, obtaining the $p_1, p_2, \ldots$ such that $H_1(pw_1) = \mathfrak{g}^{p_1}, H_1(pw_2) = \mathfrak{g}^{p_2}, \ldots$. This allows the adversary to perform an offline dictionary attack on each user with a single Send query as follows. The adversary begins a session with a target, and receives a group element U; they respond with a group element $\mathfrak{g}^x$, for a random $x \leftarrow_\$ \mathbb{Z}_p$; after receiving a message encrypted under the session key, they can check if the session key equals U^{x/p_i} for one of the p_i values, enabling password recovery.

This is prevented by having unique session identifiers for each session. In this case, the set of candidate generators for each session is unique, so the discrete logarithm computations do not carry over from one session to another. There remains an interesting question on what happens if the session identifier is not necessarily unique, but the channel identifier is. If this happens, then for any given pair of users, the channel identifier, and thus the set of candidate generators, is unique. But this pair of users always uses the same password in all of their sessions. Thus, an adversary's precomputation advantage would be restricted to one pair of users. In our analysis, however, we consider the case where the session identifiers are all unique.

6.3.3 Hashing Session Keys

In CPace$_{\text{base}}$, the session key is computed as $sk \leftarrow H_2(\mathsf{sid}||K||\textsc{oc}(U,V))$. The BPR model (described in full in Section 6.6.1) allows an adversary to obtain session keys by the Reveal query. While such a query marks that session as 'not fresh', and invalidates it for a test challenge, other sessions involving the same parties can still be considered fresh.

This makes the hashing of K critical for the quantum annoying property. If instead, the Reveal query were to provide K itself to the adversary the scheme can be broken with a single discrete logarithm. An adversary can compute $\mathsf{DLOG}(U,K)$ to recover v, then compute $V^{v^{-1}}$ to recover g_{pw} and perform an offline dictionary attack to obtain the password.

It is perhaps not surprising that hashing the Diffie–Hellman completion to generate the session key improves the security of the scheme, but it does show a clear difference between what is possible for a classical vs. quantum adversary. When a classical adversary obtains K it does not immediately provide an offline dictionary attack. For a password guess pw, trying to determine if g_{pw}, U, V, K is a valid Diffie–Hellman quadruple is exactly the decisional Diffie–Hellman problem.

For an implementor interested in making the scheme as quantum annoying as possible this is an important lesson: hash K and eliminate it from memory as soon as possible to minimize an attack surface area.

6.4 Generic group model proof of CPace$_{\text{core}}$

We now define the CPace$_{\text{core}}$ game, and prove an upper-bound on winning this game in the generic group model. The CPace$_{\text{core}}$ game is highly customized to go hand-in-hand with the task of proving security of the $\mathrm{CPace}_{\mathrm{base}}$ protocol, but the basic idea of adding a discrete logarithm oracle to the generic group model as a way to capture quantum-annoyingness may have applications beyond this specific scenario.[1]

6.4.1 CPace Core game definition

Overview. The game takes place over a collection of instances, each indexed by an integer i. For each instance i, there are N generators $g_{i,j}$. One of these generators is picked at random (represented by a target index t_i), and a Diffie–Hellman session is initiated, picking random integers $u_i, v_i \leftarrow_{\$} \mathbb{Z}_p$, and calculating $U_i \leftarrow g_{i,t_i}^{u_i}$, $V_i \leftarrow g_{i,t_i}^{v_i}$. All of this is set up by calls to a $\mathsf{NewInstance}$ oracle. We keep track of a counter variable ctr that is incremented every time $\mathsf{NewInstance}$ is called to keep track of the number of instances. When $\mathsf{NewInstance}$ is called, the adversary can optionally provide an index $\ell \leq \mathsf{ctr}$. This indicates that they want the new instance to be *linked* to a previous instances; linked instances use the same target index t_i. When we interface with a PAKE adversary, this will represent sessions being instantiated with the same password, since they are between the same parties. Note that even though the index is repeated, the set of generators is distinct.

At the beginning of the game, a challenge bit s is drawn uniformly. Eventually the adversary may call a $\mathsf{Challenge}$ oracle with an instance i, a group element W, and a bit b indicating if they want to challenge the U half or the V half. If the challenge bit $s = 0$, then we provide $H(i, W^{u_i}, \textsc{oc}(U_i, W))$ or $H(i, W^{v_i}, \textsc{oc}(V_i, W))$ depending on which half the adversary chose to challenge. If the challenge bit $s = 1$, then the response they receive is drawn uniformly from the set of confirmation values instead. The adversary is allowed to query $\mathsf{Challenge}$ twice per instance, once each for the U and V halves. The main challenge of the adversary is to determine the challenge bit s by trying to figure out the Diffie–Hellman completion without knowing which target index was used.

The interface with the $\mathsf{Challenge}$ oracle may seem somewhat arbitrary at first, with two Diffie–Hellman halves provided, and then the adversary allowed to use them separately when querying the $\mathsf{Challenge}$ oracle. When we interface with a real PAKE adversary in our

[1]We initially started out with a much simpler game in the generic group model with a discrete logarithm oracle, and planned to put most of the complexity into the AKE proof. However, as we developed the AKE proof, we frequently encountered steps where the only way we could see to proceed was to extend the generic group model game. Interestingly, the *proof* of the generic group model game often did not change very much as a result: the core idea of the proof—maintaining a linear system and checking for certain events based on the rank of a consistency matrix—was robust for the many features we added to the $\mathrm{CPace}_{\mathrm{core}}$ problem.

proof of CPace$_{\text{base}}$, this simply reflects the fact that some sessions may have one or both endpoints not controlled by the adversary.

The adversary has access to a few other sources of information. The group operation $(\cdot)$ and DLOG oracles are how the adversary can find new information about group elements and the relationships between them. The GetGen oracle gives the adversary a representation of a generator for an instance and index, and the $\mathsf{GetTarget}$ oracle tells the adversary the target index for an instance i. In order to not make the game trivial, when $\mathsf{GetTarget}$ is called, we change the behaviour of the oracle H, so that whatever information the adversary was provided before is made to be consistent with H.

Details. For a positive integer m, $[m]$ represents the integers 1 through m. If $m = 0$ it represents the empty set. Define the set $\mathcal{G} \subseteq \{0,1\}^n$ to be the representation of group elements provided to an adversary, for some suitably large n. Define $\mathcal{C} = \{0,1\}^\lambda$ to be a set of confirmation values.

Parameters of the game are N, the size of the password space and thus the number of relevant generators; and p, the (prime) size of the group. The state of the game is maintained by a non-negative integer ctr and a bit s, with ctr initially set to 0 and s sampled uniformly from $\{0,1\}$. The adversary is given (a representation of) a generator $\mathfrak{g}$ of $\mathcal{G}$, as well as the identity element. The adversary has access to the following oracles:

- $\cdot : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$: The group operation oracle.
- $\mathsf{DLOG} : \mathcal{G} \times \mathcal{G} \to \mathbb{Z}_p$: A discrete logarithm oracle.
- $H : [\mathsf{ctr}] \times \mathcal{G} \times \mathcal{G} \times \mathcal{G} \to \mathcal{C}$: A confirmation value oracle. This acts as a random oracle, taking in a counter, a Diffie–Hellman completion K, and the ordered concatenation of two group elements, and returns a uniformly random confirmation value.
- $\mathsf{GetGen} : [\mathsf{ctr}] \times [N] \to \mathcal{G}$: On input (i, j), returns $g_{i,j}$.
- $\mathsf{NewInstance} : [\mathsf{ctr}] \cup \{\perp\} \to \mathcal{G} \times \mathcal{G}$: This oracle creates a new instance of the problem. If the input is $\perp$, a new instance independent from all previous instances is generated:
    1. Increment ctr.
    2. Sample fresh generators $g_{\mathsf{ctr},j} \leftarrow\$\, \mathcal{G}$ for $j \in [N]$.
    3. Sample a uniform target index $t_{\mathsf{ctr}} \leftarrow\$\, [N]$.
    4. Sample uniform $u_{\mathsf{ctr}}, v_{\mathsf{ctr}} \leftarrow\$\, \mathbb{Z}_p$ and compute $U_{\mathsf{ctr}} \leftarrow g_{\mathsf{ctr},t_{\mathsf{ctr}}}^{u_{\mathsf{ctr}}}$, $V_{\mathsf{ctr}} \leftarrow g_{\mathsf{ctr},t_{\mathsf{ctr}}}^{v_{\mathsf{ctr}}}$.

5. Return $U_{\mathsf{ctr}}, V_{\mathsf{ctr}}$.

If the input is $\ell \leq \mathsf{ctr}$, the instance has the same target index as instance ℓ. The same steps are repeated but the same target index as that of session ℓ is used: step 3 is replaced by $t_{\mathsf{ctr}} \leftarrow t_\ell$. This instance is said to be *linked* to instance ℓ, as well as all other instances that instance ℓ is linked to.

- $\mathsf{Challenge} : [\mathsf{ctr}] \times \{0,1\} \times \mathcal{G} \to \mathcal{C}$: On input $(i, b, W_{i,b})$, if $b = 0$ we calculate $K \leftarrow W_{i,0}^{u_i}$, and if $b = 1$, $K \leftarrow W_{i,1}^{v_i}$. If K is equal to the identity element, return $\perp$. Otherwise if the challenge bit $s = 0$ or $\mathsf{GetTarget}$ has been called on this or a linked instance, then return $H(i, K, \textsc{oc}(U_i, W_{i,b}))$ or $H(i, K, \textsc{oc}(V_i, W_{i,b}))$ depending on b. If $s = 1$ and $\mathsf{GetTarget}$ has not been called on a linked instance, return a randomly sampled $h_i \leftarrow_{\$} \mathcal{C}$. This oracle can only be called twice per instance i, once with $b = 0$ and once with $b = 1$.

- $\mathsf{GetTarget} : [\mathsf{ctr}] \to [N]$: Returns the target index t_i for instance i. If $s = 1$, then for each instance linked to instance i, we reprogram H to behave correctly: modify H so that $H(i, W_{i,0}^{u_i}, \textsc{oc}(U_i, W_{i,0})) = h_{i,0}$, and $H(i, W_{i,1}^{v_i}, \textsc{oc}(V_i, W_{i,1})) = h_{i,1}$, where $h_{i,b}$ is the value that was previously provided for the challenge. If $\mathsf{Challenge}$ has not yet been called for one of the linked instances, but later is, H will skip the check for the value of s and always return the output determined by H (so that responses are consistent).

The adversary wins if any of three conditions is met:

1. The adversary queries $H(i, W_{i,0}^{u_i}, \textsc{oc}(U_i, W_{i,0}))$ after making a $\mathsf{Challenge}(i, 0, W_{i,0})$ query, but before making a $\mathsf{GetTarget}$ query on a linked instance.
2. The adversary queries $H(i, W_{i,1}^{v_i}, \textsc{oc}(V_i, W_{i,1}))$ after making a $\mathsf{Challenge}(i, 1, W_{i,1})$ query, but before making a $\mathsf{GetTarget}$ query on a linked instance.
3. At the end of the game, the adversary guesses s correctly.

We want to determine the probability of the adversary's success in terms of the number of queries they make. We count the number of queries as follows:

- $q_{\mathcal{G}}$, the number of queries to the group operation oracle.
- q_D, the number of queries to the discrete logarithm oracle.

- q_N, the number of queries to NewInstance (i.e., the total number of instances).
- q_C, the number of queries to Challenge where the adversary did *not* submit $(i, 0, V_i)$ or $(i, 1, U_i)$. In other words, the number of instances for which the adversary actively participated in the Diffie–Hellman session, rather than passively observed one.
- q_G, the number of queries to GetGen.

While there are three conditions under which the adversary wins, in truth, there is only one event that leads them to gaining an advantage in winning. The only way to find information on the challenge bit s is to detect if the output of H is correct or not for a given instance. If a GetTarget query is made, then the output of H changes to no longer depend on s for that or any linked instance, and so the relevant query must be made prior to a GetTarget query. Thus the advantage of the adversary is entirely quantified by their ability to query Challenge$(i, b, W_{i,b})$ and then either $H(i, W_{i,0}^{u_i}, \textsc{oc}(U_i, W_{i,0}))$ or $H(i, W_{i,1}^{v_i}, \textsc{oc}(U_i, W_{i,1}))$ before ever making a GetTarget(i) query.

The heart of the proof comes from the fact that even though NewInstance gives the adversary $U_i = g_{i,t_i}^{u_i}$ and $V_i = g_{i,t_i}^{v_i}$, it does not actually leak any information about what index t_i was used for instance i. We can write the elements of G in terms of the generator provided to the adversary, $\mathfrak{g}$. The N generators for instance i can be rewritten as

$$g_{i,1} = \mathfrak{g}^{p_{i,1}}, g_{i,2} = \mathfrak{g}^{p_{i,2}}, \ldots, g_{i,N} = \mathfrak{g}^{p_{i,N}}.$$

In this view, choosing a random generator corresponds to setting $U_i = \mathfrak{g}^{p_{i,t_i} \cdot u_i}$, for a random u_i and t_i. But note that *each* generator and corresponding p_{i,t_i} value is equally possible, as $p_{i,t_i} \cdot u_i = p_{i,j} \cdot (p_{i,j}^{-1} p_{i,t_i} u_i)$. Thus the only way for the adversary to proceed is to guess the generator, compute the $W_{i,0}^{u_i}$ value and query it to H. However each guess requires the adversary to know the discrete logarithm of either U_i, V_i, or $W_{i,b}$ with respect to the generator $g_{i,j}$. This requires either a discrete logarithm query to be made, or for the $W_{i,b}$ value to have been crafted so that the discrete logarithm between g_{i,t_i} and $W_{i,b}$ is known to the adversary. We will therefore establish that each query to DLOG and each customized query to Challenge essentially provides one guess for the target index t_i, so in expectation an adversary must make roughly N such queries.

Other than this, there are small terms in the upper bound that are related to the adversary finding collisions in the generators (and thus being able to make a single DLOG query relevant to multiple instances) and the adversary calculating discrete logarithms by making group operation, rather than DLOG, queries, both of which are divided by the group order p, which is cryptographically large.

Theorem 17. *Let $\mathcal{A}$ be an adversary in the* CPace$_{\text{core}}$ *game. The probability that $\mathcal{A}$ wins the game is at most*

$$\frac{1}{2} + \frac{q_D + q_C}{N} + O(q_G^2/p) + O(q_D q_G^2/p).$$

6.4.2 Proof outline

As is typical for generic group model proofs, we will maintain a table T that translates between the (additive) secret representation of elements as numbers in $\mathbb{Z}_p$ and the (multiplicative) public representation provided to the adversary, which are random unique elements of $\{0,1\}^n$. The secret representation of the identity element is 0, and the secret representation of the generator $\mathfrak{g}$ is 1.

For an instance i and bit b, let $W_{i,b}$ be the group element that the adversary submitted to the Challenge oracle for the bit b, and let $w_{i,b}$ be the discrete logarithm between g_{i,t_i} and $W_{i,b}$). To provide an upper bound on the adversary's ability to guess s, we need to determine their ability to query $g_{i,t_i}^{u_i w_{i,0}}$ or $g_{i,t_i}^{v_i w_{i,1}}$ to H. Except where it is relevant, for ease of notation, we will focus on the $b = 0$ case for the adversary's challenge queries, with the understanding that an implicit 'and similarly for $b = 1$' follows.

If $\mathfrak{g}^{p_{i,t_i}} = g_{i,t_i}$, then this would mean that the adversary would be unable to make a relevant query until the secret representation $p_{i,t_i} u_i w_{i,0}$ of $g_{i,t_i}^{u_i w_{i,0}}$ is added to T. However, rather than maintaining a specific $p_{i,j} \in \mathbb{Z}_p$ as the secret representation of $g_{i,j}$, we will instead maintain a variable $X_{i,j}$. For example, say the adversary queries $g_{1,1} \cdot \mathfrak{g}$ to the group operation oracle. With a specific $p_{1,1}$ in mind such that $g_{1,1} = \mathfrak{g}^{p_{1,1}}$, the secret representation of such an element would be $p_{1,1} + 1$. Instead, we write the secret representation as the linear combination $X_{1,1} + 1$, and, if we have not seen this linear combination before, choose a new public representation for it and return that to the adversary.

Similarly, the adversary may query $g_{1,1} \cdot g_{1,1}$ to the group operation oracle. We would record $2X_{1,1}$ in the table, and assuming that this term has not appeared before, give it a random unused representation. Other generators have corresponding variables $X_{i,j}$. By making group operation oracle queries combining these terms, arbitrary linear combinations of these variables can be added into the table T. This allows us to precisely quantify the information that the adversary has obtained through the discrete logarithm oracle, which in turn will allow us to precisely calculate the probability that the adversary is capable of causing certain events to happen, like making relevant queries to H.

On the other hand, the discrete logarithm oracle informs the adversary of the relationship between those linear combinations. For example, if the adversary has used the group operation oracle to figure out the representation of $\mathfrak{g}^c$ and $\mathfrak{g}^d$, and queries these to the discrete logarithm oracle, they must be provided with $c^{-1}d \bmod p$. Of course such a query provides no additional information to the adversary as they could compute it themselves. Useful queries to the discrete logarithm oracle involve group elements given to the adversary from the $\mathsf{NewInstance}$ or GetGen oracles. If the query $\mathsf{DLOG}(\mathfrak{g}, g_{i,j})$ is made, then a value for the corresponding $X_{i,j}$ must be decided and provided as a response.

In order to query H with the completion of the Diffie–Hellman-like session, the discrete logarithm between at least one of (g_{i,t_i}, U_i), (g_{i,t_i}, V_i), or $(g_{i,t_i}, W_{i,b})$ must be defined. If all are undefined, the completion is undefined as well, and not possible to query. In our table T, the secret representation of U_i should be $u_i X_{i,t_i}$. However, we will instead choose a constant $\mu_i \in \mathbb{Z}_p$ and set that to be the secret representation of U_i (the secret representation of V_i will be ν_i). This means that until the adversary makes a DLOG query that causes X_{i,t_i} to become defined, u_i will not be defined.

When the adversary makes a $\mathsf{Challenge}$ query for an instance i, they choose a bit b and submit a group element $W_{i,b}$. In our table T, $W_{i,b}$ will have a secret representation of some linear combination of the $X_{i,j}$ variables, plus a possible constant. We must also consider the adversary's ability to cause $\mathsf{DLOG}(g_{i,t_i}, W_{i,b})$ to become defined. Essentially, the adversary will get one guess per challenge query. The adversary can select an index j and hope that $j = t_i$. Then they can construct the challenge so that they know the $w_{i,b}$ such that $W_{i,b} = g_{i,t_i}^{w_{i,b}}$, in which case the discrete logarithm is defined and the adversary can complete the challenge. We will establish however, that the adversary will only get one such guess out of the $\mathsf{Challenge}$ queries that they craft themselves to try to make the discrete logarithm defined.

So, the overall idea of the proof is that queries to the group operation oracle populate the table T with linear expressions and the discrete logarithm oracle enforces linear relationships between those expressions. With enough queries to the discrete logarithm oracle, the adversary can force enough relations between the various $X_{i,j}$ values that each one is entirely decided. But unless the value of X_{i,t_i} has been defined by making the proper queries to the discrete logarithm oracle, or the adversary manages to guess t_i when making a challenge oracle query, there is no way for the adversary to query $g_{i,t_i}^{u_i w_{i,b}}$ to H, as that value is undefined, and thus has not been given a public representation.

Each query to the DLOG oracle imposes at most one linear constraint on the $X_{i,j}$ variables. Since any given j is not more likely than any other from the adversary's perspective, we need to consider the expected number of linear constraints that need to be put on the X variables before X_{i,t_i} is defined. We will show that the probability X_{i,t_i} is defined after q_D queries to the discrete logarithm oracle is at most q_D/N, which corresponds exactly to picking one session and performing a brute force search of computing the discrete logarithm of $g_{i,1}, g_{i,2} \dots$. (Viewed as a PAKE, this matches the quantum annoying property exactly: the adversary guesses the password, computes the generator that corresponds to that password, and finds the discrete logarithm with respect to that generator to make a guess towards the secret key.)

The remaining terms in the theorem's bound, $O((q_D q_{\mathcal{G}}^2 + q_G^2)/p)$ come from the adversary's ability to distinguish that the oracle has been managed with unknown $X_{i,j}$ variables, rather than 'real' secret representations. The $O(q_G^2/p)$ term comes from the fact that we will provide each generator with a unique representation, while in the real world, we would expect there to eventually be collisions in the representation of the generators.

The numerator in the other term, $q_D q_{\mathcal{G}}^2$, is asymptotically the same as the number of queries to the group operation oracle required to calculate a discrete logarithm (e.g., using the baby-step giant-step algorithm). So in our model, if the adversary uses the group operation oracle to calculate a discrete logarithm, rather than the provided discrete logarithm oracle, then they may notice that the discrete logarithm oracle is not behaving entirely faithfully. This happens because, when calls to the discrete logarithm oracle are made, the values of the $X_{i,j}$ can become defined. If enough group elements have been added to the table T, then it is possible that when an $X_{i,j}$ becomes defined, two of the linear polynomials in T will take on the same value in $\mathbb{Z}_p$, even though the adversary was given different representations in the generic group. However for large p, roughly $\sqrt{p}$ values need to be added to the table T in order to expect a collision to occur (the birthday paradox has come into effect).

Hence, the adversary's advantage in distinguishing how the discrete logarithm and group operation oracles are managed grows as $O((q_G + q_D q_{\mathcal{G}}^2)/p)$.

6.5 Proof of Theorem 17

We now get into the specifics of the proof: how is the table T managed, what exactly are the linear relations imposed by the discrete logarithm oracle, and proofs of the bounds. Algorithms in Figures 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, and 6.8 provide a reference for how all of the algorithms are simulated.

The main technical points of the proof consist of: how group operation oracle queries add entries to T, how the NewInstance, GetGen, and Challenge oracles allow the adversary to begin interacting with the group, how discrete logarithm oracle queries are answered, and how we guarantee that responses to the oracles are consistent with each other and with

```
Simulating New Instance queries in GGM

     Input: Integer $\ell \leq \mathsf{ctr}$ or $\perp$
1 :  Increment $\mathsf{ctr}$
2 :  if input was integer $\ell$ then
3 :    Set the target index $t_{\mathsf{ctr}} \leftarrow t_\ell$
4 :    Mark instance $\mathsf{ctr}$ as linked to instance $\ell$ as well as all instances $\mathsf{ctr}$ is
       linked to, and vice versa.
5 :  else
6 :    Sample a uniform $t_{\mathsf{ctr}} \leftarrow\!\!\$\ [N]$
7 :  endif
8 :  Sample a uniform $\mu_{\mathsf{ctr}}, \nu_{\mathsf{ctr}} \leftarrow\!\!\$\ \mathbb{Z}_p^2$
9 :  Sample public representations $U_{\mathsf{ctr}}, V_{\mathsf{ctr}}$
10 : Add $(\mu_{\mathsf{ctr}}, U_{\mathsf{ctr}})$ and $(\nu_{\mathsf{ctr}}, V_{\mathsf{ctr}})$ to table $T$return $U_{\mathsf{ctr}}, V_{\mathsf{ctr}}$.
```

Figure 6.2: Simulating New Instance queries in the generic group model.

past responses. With this in hand we bound the probability that the discrete logarithm between g_{i,t_i} and U_i, V_i, or $W_{i,b}$ is defined after q_D queries to the discrete logarithm oracle and q_C modified queries to the Challenge oracle.

Simulating DLOG queries in GGM

```
    Input: Query $(g_a, g_b) \in \mathcal{G} \times \mathcal{G}$, table $T$, matrix $D$, and row $\vec{r}$.
1 : Use table $T$ to look up secret representations $g_a \hookrightarrow a_0 + \sum_{i,j} a_{i,j} X_{i,j}$
     and $g_b \hookrightarrow b_0 + \sum_{i,j} b_{i,j} X_{i,j}$
2 : if either secret representation doesn't appear in $T$ then return $\perp$ endif
3 : Select a uniform $\vec{s}$ such that $D\vec{s} = \vec{r}$.
4 : Compute $\delta = (\vec{a} \cdot \vec{s})^{-1}(\vec{b} \cdot \vec{s})$.
5 : Compute the row $\delta\vec{a} - \vec{b}$ and the value $b_0 - \delta a_0$
6 : Append the row to $D$ and the value to $\vec{r}$.
7 : return $\delta, D, \vec{r}$.
```

Figure 6.3: Simulating DLOG queries in the generic group model.

Simulating Challenge queries in GGM

```
    Input: Instance $i \in [\mathsf{ctr}]$, bit $b \in \{0,1\}$, group element $W_{i,b} \in \mathcal{G}$.
1 : if challenge query starting with $i$ and $b$ has been made before then return $\perp$ endif
2 : if Instance $i$ or a linked instance has had a GetTarget query issued then
3 :   if $b = 0$ then return $H(i, W_{i,b}^{u_i}, oc(U_i, W_{i,b}))$
4 :   else return $H(i, W_{i,b}^{v_i}, oc(V_i, W_{i,b}))$
5 :   endif
6 : else
7 :   $h_{i,b} \leftarrow\$ \mathcal{C}$, return $h_{i,b}$
8 : endif
```

Figure 6.4: Simulating Challenge queries in the generic group model.

Simulating GetGen queries in GGM

```
    **Input:** Instance $i \in [\mathsf{ctr}]$, index $j \in [N]$, table $T$
1 : if $X_{i,j}$ already appears in the table $T$ then
2 :   return Corresponding public representation $g_{i,j}$.
3 : else
4 :   Sample a new public representation $g_{i,j}$
5 :   Add $(X_{i,j}, g_{i,j})$ to the table $T$
6 :   return $g_{i,j}$
7 : endif
```

Figure 6.5: Simulating GetGen in the generic group model.

Simulating GetTarget queries in the GGM

```
Input: Instance $i \in [\mathsf{ctr}]$.
if GetTarget has never been called before on $i$ or a linked instance then
  for each instance $j$ linked to instance $i$ (including $i$) do
    Mark that instance has had a GetTarget query called on a linked instance.
    Query $\mathsf{DLOG}(g_{j,t_j}, U_j)$ and $\mathsf{DLOG}(g_{j,t_j}, V_j)$ to cause $X_{j,t_j}, v_{j,t_j}$ to become
    defined (if not already defined).
    Calculate $u_j \leftarrow X_{j,t_j}^{-1}\mu_j,\ \ v_j \leftarrow X_{j,t_j}^{-1}\nu_j$.
    if Challenge$(j, 0, W_{j,0})$has been called then
      if $H(i, W_{j,0}^{u_j}, oc(U_j, W_{j,0}))$ has been called then
        Adversary has won game, abort.
      else
        Program oracle $H$ so that $H(i, W_{j,0}^{u_j}, oc(U_j, W_{j,0}))$ returns $h_{j,0}$, the
        response to the Challenge query.
      endif
    endif
    if Challenge$(j, 1, W_{j,1})$ has been called then
      if $H(i, W_{j,1}^{v_j}, oc(V_j, W_{j,1}))$ has been called then
        Adversary has won game, abort.
      else
        Program oracle $H$ so that $H(i, W_{j,1}^{v_j}, oc(V_j, W_{j,1}))$ returns $h_{j,1}$, the
        response to the Challenge query.
      endif
    endif
  endfor
endif
return $t_i$
```

Figure 6.6: Simulating GetTarget queries in the generic group model.

```
Simulating group operation oracle queries in GGM

Input: query $(g_a, g_b) \in \mathcal{G} \times \mathcal{G}$, table $T$, matrix $D$, and row $\vec{r}$.
1 : Use table $T$ to look up secret representations $g_a \hookrightarrow a_0 + \sum_{i,j} a_{i,j} X_{i,j}$
    and $g_b \hookrightarrow b_0 + \sum_{i,j} b_{i,j} X_{i,j}$.
2 : if either secret representation doesn't appear in $T$ then return $\perp$ endif
3 : if $C(\vec{X}) = a_0 + b_0 + \sum_{i,j}(a_{i,j} + b_{i,j}) X_{i,j}$ appears in $T$ then
4 :   Return corresponding secret representaiton.
5 : else
6 :   for each secret representation $F(\vec{X})$ in table $T$ do
7 :     Compute row $\vec{g} = [c_{1,1} - f_{1,1}, c_{1,2} - f_{1,2}, \ldots, c_{q_N,N} - f_{q_N,N}]$
      and value $e = f_0 - c_0$.
8 :     if $\vec{g}$ is linearly independent from the rows of $D$ then
9 :       continue
10 :    else
11 :      Find the $\vec{h}$ such that $\vec{h} \cdot D = \vec{g}$.
12 :      if $\vec{h} \cdot \vec{r} = -e$ then
13 :        return public representation of $F(\vec{X})$.
14 :      endif
15 :    endif
16 :  endfor
17 :  Sample a new public representation $g_c$ for $C(\vec{X})$.
18 :  Add $C(\vec{X}), g_c$ to the table $T$.
19 :  return $g_c$
20 : endif
```

Figure 6.7: Simulating group operations in the generic group model.

```
Simulating $H$ queries in GGM
    **Input:** Instance $i \in [\mathsf{ctr}]$, group element $K \in \mathcal{G}$,
    ordered group elements $A, B \in \mathcal{G}^2$, hash table
1 : if Query has been previously made or programmed then
2 :     return the same response $h$.
3 : else    Sample $h \leftarrow\$ \mathcal{C}$
4 :     Record query and $h$ into hash table
5 :     return $h$
6 : endif
```

Figure 6.8: Simulating the random oracle in the generic group model.

The group operation oracle and the table T. The table T is used to convert between the public representations provided to the adversary and the secret representation of the element in the additive group $\mathbb{Z}_p$. To begin with, the table has just 2 elements in it: a generator $\mathfrak{g}$ and the identity element. The public representation of each of these elements is chosen at random from $\{0,1\}^n$. Note that it is common in generic group model proofs to choose n large enough so that we do not need to worry about collisions in our representations, or the adversary 'guessing' a group element that has not been added to T. Since new public representations are added to T by queries to the group operation and GetGen oracles, choosing $n \gg \log_2(q_{\mathcal{G}} + q_G + p)$ is sufficient. It is also easy to check and see if a representation has already been used and, if so, re-sample. Since it is easy to choose a large enough n, and it impacts no other parts of the proof, we omit a term that considers the probability of picking the same representation twice.

The secret representation of $\mathfrak{g}$ is naturally 1, the identity element 0, and the secret representation of each $g_{i,j}$ from GetGen is represented by a variable $X_{i,j}$. When the group operation oracle is queried on elements g_a and g_b, the public representations are queried in the table to find the corresponding secret representation. If no such representation exists, then the query is considered invalid, and returned as such[2]. Otherwise, the secret representation of g_a and g_b will be two linear combinations of the $X_{i,j}$ variables as well as a possible constant, which we can write as $g_a \hookrightarrow a_0 + \sum_{i\in[q_N]}\sum_{j\in[N]} a_{i,j}X_{i,j} = a_0 + \vec{a}\cdot\vec{X}$ and $g_b \hookrightarrow b_0 + \sum_{i\in[q_N]}\sum_{j\in[N]} b_{i,j}X_{i,j} = b_0 + \vec{b}\cdot\vec{X}$, with $a_{i,j}, b_{i,j} \in \mathbb{Z}_p$. We can then compute the secret representation of $g_a \cdot g_b \hookrightarrow (a_0 + b_0) + \sum_{i,j}(a_{i,j} + b_{i,j})X_{i,j}$. Once the secret representation has been computed, we can check to see if this new linear combination already exists in the table. If it does, then use the already existing representation of the group element. If not, then we can generate a new random public representation for this new linear combination and provide it to the adversary.

This simple check will only work until the adversary begins to make discrete logarithm oracle queries. As queries to the discrete logarithm oracle impose linear relationships between the $X_{i,j}$ variables, we need to check if that linear combination *modulo the relations defined* already exists in the table. We will discuss more on this point when we explain how linear relationships between the $X_{i,j}$ variables are defined by queries to the discrete logarithm oracle. As a preview, these linear relations will be encoded into a matrix D. To check and see if two secret representations $\vec{a}$ and $\vec{b}$ actually encode the same group element, we see if $\vec{a} - \vec{b}$ is linearly independent from the rows of D. If it is, then its value is not

[2]This is correct behaviour so long as the representation does not later become valid. Since representations are randomly chosen, the probability that this happens is negligible in n, the bit length of the representations. As discussed, we assume n is chosen to make this probability negligible.

dependent on the linear relations that have been defined, and we can conclude that these represent distinct group elements.

Note as well that the group operation oracle can be extended to allow for inverses to be calculated as well. This simply means calculating $-a_0 - \sum_{i\in[q_N]} \sum_{j\in[N]} a_{i,j} X_{i,j}$ and otherwise performing the same sequence of steps.

Oracles $\mathsf{NewInstance}$, GetGen, and $\mathsf{Challenge}$. The game begins with the adversary only aware of a single generator element and the identity element. In order to begin meaningfully interacting with the game, $\mathsf{NewInstance}$ must be called. When this happens, we increment ctr, and if the instance is not linked to another instance, then we sample a new target index t_{ctr} from $[N]$.

Rather than earnestly generating a Diffie–Hellman-like instance from $g_{\mathsf{ctr},t_{\mathsf{ctr}}}$, we instead sample values $\mu_{\mathsf{ctr}}, \nu_{\mathsf{ctr}} \leftarrow_{\$} \mathbb{Z}_p$. We will set $U_{\mathsf{ctr}} \leftarrow \mathfrak{g}^{\mu_{\mathsf{ctr}}}$, $V_{\mathsf{ctr}} \leftarrow \mathfrak{g}^{\nu_{ctr}}$. We calculate the public representation of these elements (which may be entirely new, requiring new entries into T), and return the public representation to the adversary.

We do this, rather than sending honestly generated U_i and V_i values in order to allow the discrete logarithm between g_{i,t_i} and U_i or V_i to remain undefined. Note that this does not affect the distribution of U_i or V_i. Since u_i and v_i are chosen uniformly at random, choosing the products μ_{ctr} and ν_{ctr} uniformly matches the distribution exactly. But until X_{i,t_i} becomes defined, the discrete logarithm between g_{i,t_i} and U_i and V_i is similarly undefined.

After having created an instance, the adversary can access generators through the GetGen oracle. When $\mathsf{GetGen}(i,j)$, where $i \leq \mathsf{ctr}$, is called, we sample a new public representation and add it and $X_{i,j}$ to T. We always sample unique representations, and this does create a small incongruity with the real game. In the real game, after sampling roughly $\sqrt{p}$ generators, an adversary would expect to see repetition in the public representations. But we will always provide unique representations, no matter how many times the oracle is called. This results in a $O(q_G^2/p)$ term in the theorem statement, representing the adversary's ability to cause a collision in the generators.

The challenge oracle is how the adversary is able to gain an advantage in winning the game. When $\mathsf{Challenge}(i, 0, W_{i,0})$ is called, we are expected to respond with either a random $h_i \leftarrow_{\$} \mathcal{C}$ or $H(i, W_{i,0}^{u_i}, oc(U_i, W_{i,0}))$. We will always respond with a random h_i, so long as $\mathsf{GetTarget}(i)$ has not been called on a related instance i. This is indistinguishable as long as the adversary does not query $W_{i,0}^{u_i}$ without having previously made a $\mathsf{GetTarget}(i)$ query. If such a query is made, we consider them to have won.

The discrete logarithm oracle and the linear relationship matrix D. For queries to the discrete logarithm oracle, we need to define what linear relations are imposed, and how future oracle responses are managed for consistency. When group elements with secret representations α and $\beta \in \mathbb{Z}_p$ are queried, the response should be a value $\delta \in \mathbb{Z}_p$ such that $\alpha \cdot \delta \equiv \beta \pmod{p}$. So when a group element with secret representation $\alpha = a_0 + \sum a_{i,j} X_{i,j}$ and $\beta = b_0 + \sum X_{i,j}$ are queried, by returning a value $\delta \in \mathbb{Z}_p$ we are declaring that

$$\delta\big(a_0 + \sum_{i,j} a_{i,j} X_{i,j}\big) = b_0 + \sum_{i,j} b_{i,j} X_{i,j},$$

or equivalently,

$$\sum_{i,j} (\delta a_{i,j} - b_{i,j}) X_{i,j} = b_0 - \delta \cdot a_0. \tag{6.1}$$

When a discrete logarithm oracle query is made, we thus need to choose a value δ consistent with all previous δ values provided. To do this we maintain a matrix D and a vector $\vec{r}$ that encodes all previous responses. That is, when a linear equation (6.1) is defined, we append the row

$$[\delta \cdot a_{1,1} - b_{1,1} \quad \delta \cdot a_{1,2} - b_{1,2} \quad \dots \quad \delta \cdot a_{q_N,N} - b_{q_N,N}] \tag{6.2}$$

to D and extend $\vec{r}$ by the entry $b_0 - \delta \cdot a_0$. Thus the set of responses provided to the adversary so far imposes the linear constraints $D\vec{X} = \vec{r}$, where

$$\vec{X} = [X_{1,1}, X_{1,2}, \dots, X_{1,N}, X_{2,1}, \dots, X_{q_N,N}]^T.$$

With this linear system in place, when a new query comes in, we can pick an arbitrary $\vec{s}$ such that $D\vec{s} = \vec{r}$, i.e., an arbitrary solution. This can be done by, for example, finding one solution and then choosing an arbitrary point in the kernel of D. Then to respond to the query $(a_0 + \vec{a} \cdot \vec{X}, b_0 + \vec{b} \cdot \vec{X}$, we can replace the $X_{i,j}$ values with the random $s_{i,j}$ values, and respond with $\delta = (a_0 + \vec{a} \cdot \vec{s})^{-1}(b_0 + \vec{b} \cdot \vec{s})$. We then add the row from (6.2) to D and append $b_0 - \delta \cdot a_0$ to $\vec{r}$. Our new answer is guaranteed to be consistent with all previous responses as it is consistent with $\vec{s}$, which was chosen from the solution space.

This also allows us to tell if a given $a_0 + \vec{a} \cdot \vec{X}$ has a value determined by D and $\vec{r}$, and if so, what that value is. If we can construct a linear combination of the rows of D that add up to $\vec{a}$, then the value of the linear combination is determined. Let $\vec{w}$ be the linear combination of rows, so that $\vec{w}^T D = \vec{a}^T$. Then the matrix D is telling us that $\vec{a} \cdot \vec{X} = (\vec{w}^T D)\vec{X} = \vec{w}^T(D\vec{X}) = \vec{w}^T \vec{r} = \vec{w} \cdot \vec{r}$. Thus the value (in $\mathbb{Z}_p$) of $a_0 + \vec{a} \cdot \vec{X}$ is $a_0 + \vec{w} \cdot \vec{r}$, where $\vec{w}$ is the linear combination of the rows of D that add up to $\vec{a}$. If there is no such linear combination, i.e., $\vec{a}$ is not in the rowspace of D, then the value of $a_0 + \vec{a} \cdot \vec{X}$ is not yet determined by D and $\vec{r}$. When D and $\vec{r}$ do determine a secret representation's value in $\mathbb{Z}_p$, we will write it as $\cong$. So if $\vec{w}^T D = \vec{a}^T$, then $a_0 + \vec{a} \cdot \vec{X} \cong a_0 + \vec{w} \cdot \vec{r}$.

Now we may discuss how we check if a linear combination modulo the linear constraints has already appeared in the table T. When a group operation oracle query is made that will add the secret representation $a_0 + \vec{a} \cdot \vec{X}$ to the table T, we consider the difference $\vec{a} - \vec{b}$ between $\vec{a}$ and the coefficients of every other linear combination of $X_{i,j}$ values in the table T, $\vec{b}$. For each difference $\vec{a} - \vec{b}$ we need to check to see if the linear relations set forth by D mean that the new group element $a_0 + \vec{a} \cdot \vec{X}$ is actually the same as $b_0 + \vec{b} \cdot \vec{X}$.

To do this, we check to see if the rank of the matrix D is increased by appending $\vec{a} - \vec{b}$ as a row. If the rank does increase, this tells us that the relation between $\vec{a}$ and $\vec{b}$ is not defined by D. But if the rank does not increase, then the relation is defined. This means we can find the value $c \in \mathbb{Z}_p$ such that $\vec{a} - \vec{b} \cong c$.

If $c = b_0 - a_0$, then we know that these two group elements with secret representations $a_0 + \vec{a} \cdot \vec{X}$ and $b_0 + \vec{b} \cdot \vec{X}$ must be the same given the relations provided to the adversary by the discrete logarithm oracle. In this case, a new entry does not need to be added to the table T, and instead the public representation for the element already provided can be given. If $c \neq b_0 - a_0$, then the matrix D is telling us that $\vec{a}$ and $\vec{b}$ differ by a constant factor, but they are not the same element, and so the next $\vec{b}$ can be checked.

One counterintuitive aspect is that the group operation oracle is being simulated in a very expensive way. Each time a query is made, the simulator checks against each previous query made, resulting in quadratic expense. But this is not relevant to the bounds in the proof. We are not reducing $\text{CPace}_{\text{core}}$ to another problem, but providing an information-theoretic bound in terms of the number of oracle calls being made. Thus, it does not matter how efficient the simulator is, only that it counts the number of queries to the various oracle properly.

At this point, we have guarantees that (i) when a response to a discrete logarithm query is provided, it is consistent with all previous responses to the discrete logarithm oracle, and (ii) when a response to a group operation query is provided, it is consistent with all previous responses to both the group operation oracle and the discrete logarithm oracle. The remaining question is whether responses to the discrete logarithm oracle are consistent with the previous responses to the group operation oracle. In fact, they are not guaranteed to be so. Consider the case where the adversary enumerates through the entire group to get the representation of $\mathfrak{g}, \mathfrak{g}^2, \mathfrak{g}^3, \ldots, \mathfrak{g}^{p-1}$. These will all be given different public representations, but the representations will also be different from those given to all of the generators returned from GetGen. If a discrete logarithm query of the form $(\mathfrak{g}, g_{i,j})$ is made, a specific $p_{i,j} \in \mathbb{Z}_p$ will be provided. But we will have already given g^{p_i} a different representation than $g_{i,j}$, causing an inconsistency.

Since the discrete logarithm oracle responds with random answers from the solution space, these inconsistencies require the adversary to make an enormous number of group operation oracle queries to happen: it is only if $O(\sqrt{p})$ queries to the group operation oracle occur that we must worry about this inconsistency. We provide a full justification for this claim after briefly discussing the adversary's success probability.

We now return to the analysis of this game: what is the probability that the adversary succeeds after making discrete logarithm queries, and what is the difference between managing the group operation and discrete logarithm oracles in this way and a 'proper' way?

As discussed, the adversary must have done one of two things in order to possibly win. For a session i not linked to a session where a $\mathsf{GetTarget}$ query has been made, they must either know the discrete logarithm between g_{i,t_i} and either U_i, V_i, or $W_{i,b}$. We need to characterize when it is possible for an adversary to learn this based on the matrix D, and then provide an upper bound on the adversary's success probability in triggering that event.

Lemma 29. *Let $\vec{e}_{i,t_i}$ be the standard basis vector in $\mathbb{Z}^{q_N \cdot N}$ with a 1 in position (i, t_i) and 0 everywhere else. Let $\vec{w}$ be the vector representation of $W_{i,b}$, the row vector whose entries are the coefficients of the $X_{i,j}$ variables in the corresponding secret representation. Let D be the matrix of linear relations defined by the queries to the discrete logarithm oracle. Then the discrete logarithm between g_{i,t_i} and U_i, V_i, or $W_{i,b}$ is only defined if $\vec{e}_{i,t_i}$ appears in the rowspan of $\left[\frac{D}{\vec{w}}\right]$.*

Proof. Recall that the secret representation of g_{i,t_i} is X_{i,t_i}, and for U_i and V_i it is a randomly chosen pair $\mu_i, \nu_i \leftarrow_{\$} \mathbb{Z}_p^2$. For the discrete logarithm between these two to be defined, the value X_{i,t_i} must be forced to have a specific value from the linear constraints of D. If it is not entirely constrained, then it can still take on any value in $\mathbb{Z}_p$. For it to take on a specific value, it must be the case that there is a linear combination of the rows of D that add up to $\vec{e}_{i,t_i}$.

Similarly, for $W_{i,b}$ we consider its vector representation $\vec{w}$. For the discrete logarithm to be defined, we must be able to rewrite this vector as a multiple of X_{i,t_i}. We can assume that X_{i,t_i} is undefined, since it being defined was already covered by the previous case. So, it must be possible, modulo the linear relations defined, to rewrite $\vec{w}$ as a multiple of $\vec{e}_{i,t_i}$. But 'zeroing out' the other entries of $\vec{v}$ like this means that $\vec{e}_{i,t_i}$ is in the rowspan of $\left[\frac{D}{\vec{w}}\right]$, as expected. $\square$

Corollary 7. *Let W be the matrix whose rows consist of the vectorizations of each $W_{i,b}$ submitted to the* Challenge *oracle not equal to U_i or V_i. Then the instances i for which $W_{i,b}^{u_i}$ can be queried to H are restricted to those where $\vec{e}_{i,t_i}$ appears in $\left[\frac{D}{V}\right]$.*

This corollary allows us to calculate the overall probability of having the relevant discrete logarithms defined, and thus the probability of querying a Diffie–Hellman completion and winning the game. The rank of the matrix $\left[\frac{D}{W}\right]$ is at most the number of rows of D plus the number of rows of W, which is $q_D + q_C$, the number of Challenge queries where a customized $W_{i,b}$ was submitted. The rank also limits the number of basis vectors that can appear in the row span to the same number, so at most $q_D + q_C$ basis vectors can appear there.

This is how we can bound the probability that the adversary can submit the relevant group element to H. To do this, they need to have an instance i for which no GetTarget query has been made for any linked instance, and $\vec{e}_{i,t_i}$ is in the rowspan of $\left[\frac{D}{W}\right]$. Since no GetTarget query has been made for this instance, the distribution of which $\vec{e}_{i,j}$ basis vectors appear in the rowspan is independent of t_i. Thus the adversary has $q_D + q_C$ chances for a target basis vector to appear in the rowspan. So the overall probability that one appears can be upper-bounded as $(q_D + q_C)/N$.

Next we consider the question of whether the adversary can detect that we are not managing the group operation and discrete logarithm oracles perfectly. As mentioned, we do not ensure that responses to the DLOG oracle are perfectly consistent with all previous group operation oracle queries. With enough entries in T it is possible for the adversary to notice a discrepancy in how queries were handled. For example, say the adversary has queried for group elements with secret representation X_1 and d, and in the process of making discrete logarithm queries, the value of X_1 is set to be d. Since that happens after having queried X_1 and d, the two group elements will be given different public representations.

To determine the probability that any inconsistency occurs, we consider each pair of linear combinations in the table T, $(a_0 + \vec{a} \cdot \vec{X}, b_0 + \vec{b} \cdot \vec{X})$. For a given pair, we want to check to see if a new linear constraint added to D has made these two previously distinct elements take on the same value. This occurs if, before the discrete logarithm oracle query, $\vec{a} - \vec{b}$ was linearly independent from the rows of D, but after updating to D', it is now linearly *dependent*, and furthermore we have that $\vec{a} - \vec{b} \cong b_0 - a_0$.

For every pair of elements, the probability that this happens is at most $1/p$. To see this we will discuss the geometric structure of how linear constraints are added to D and what two intersecting elements means in this geometry.

Each row of D and $\vec{r}$ adds a linear constraint to the system. If the first row of D is $\vec{d}$ and the first entry of $\vec{r}$ is r then the solution space is constrained so that $\vec{d} \cdot \vec{X} = r$. We can view this as an affine hyperplane, an $(N-1)$-dimensional subspace of $\mathbb{Z}_p^N$. When a new row is added, this corresponds to adding another affine hyperplane. The solution space is the intersection of all hyperplanes.

The process of adding a new row to D is to select a random point in the solution space, and then construct a response to the adversary's query. The adversary's query can be seen as determining the direction of the affine hyperspace (i.e., the linear subspace that goes through zero), but the response is determined by choosing a random point in the solution space and offsetting the submitted linear subspace so that it goes through that random point, constructing an affine space.

Meanwhile, pairs in our table T collide if $\vec{a}-\vec{b}$ is linearly independent before a row is added, but linearly dependent after. The vector $\vec{a}-\vec{b}$ and value $b_0 - a_0$ also can be viewed as a hyperplane H. So the geometric interpretation of the linear relations $(D, \vec{r})$ forcing $(\vec{a}-\vec{b}) \cdot \vec{X}$ to be equal to $b_0 - a_0$ is that the solution space S is contained entirely within the hyperplane H.

This case occurs if, before a new hyperplane is added to the linear constraints, the solution space is not entirely within the hyperplane H, but, after the discrete logarithm oracle query, it is. As discussed, the process of adding a new hyperplane involves picking a random point in the solution space and making sure the new hyperplane goes through that point. For the resulting solution space to be entirely within H, it must be the case that the random point that is chosen is also within H. So the question becomes, how many points in the solution space S are also in H? Since it is not the case that S is entirely contained within H, it cannot be all of them. Since S is generated by the intersection of a series of affine hyperplanes, the intersection between that and H must be either empty, or is at most a fraction $1/p$ of the space S, as desired. This is because the intersection of such hyperplanes is an affine subspace with smaller dimension. Our base field is $\mathbb{Z}_p$, and so the subspace must have a size a power of p.

So each time a new linear constraint is added to D and $\vec{r}$, for every two entries in the table T there is at most a $1/p$ chance that these two entries now represent the same group element, modulo these constraints. Since this happens for each pair in the table, we can upper bound the overall probability of any collision happening as $O(q_G^2/p)$, and the probability of a collision happening on any of the q_D queries to the discrete logarithm oracle as $O(q_D q_G^2/p)$.

Thus the probability that the adversary notices the oracles misbehaving is at most $O(q_D q_G^2/p)$, the probability that it is noticed that generators are always unique is at most $O(q_G^2/p)$, and the probability that they win assuming they do not notice misbehaviour is at most $(q_D + q_C)/N$. So the overall probability of winning is at most

$$\frac{1}{2} + (q_D + q_C)/N + O((q_D q_{\mathcal{G}}^2 + q_G^2)/p).$$

6.6 PAKE security of CPace$_{\text{base}}$

In this section we show that CPace$_{\text{base}}$ is a secure password-authenticated key exchange protocol in a variant of the Bellare–Pointcheval–Rogaway (BPR) model [23], assuming the difficulty of the CPace$_{\text{core}}$ problem from Section 6.4. Our BPR$'$ security model differs from the BPR model in that it does not provide forward secrecy, assumes a balanced PAKE (i.e., the server stores the client's password directly, not a transformation thereof), and accommodates externally specified session identifiers, in addition to providing generic group model oracles.

6.6.1 The BPR$'$ Model

Participants and passwords. Fix a non-empty finite set $\mathcal{C} \cup \mathcal{S}$ of participants; each participant is either a client or server, but never both. For each client-server pair (C, S), a password $pw_{C,S}$ is chosen uniformly at random from a set $\mathcal{P}$; each client and server has a copy of the passwords relevant to them.[3]

Sessions and state. Each participant P can execute multiple instances of the protocol simultaneously, each of which is called a session; sessions within a party are numbered sequentially, and the ith session at participant P is denoted π_P^i. For each session i, participant P maintains the following state variables:

- $\mathsf{acc}_P^i \in \{\mathsf{true}, \mathsf{false}\}$: whether the instance has successfully accepted a session key
- $\mathsf{term}_P^i \in \{\mathsf{true}, \mathsf{false}\}$: whether the instance has terminated, meaning no more incoming or outgoing messages
- state_P^i: private state of the protocol execution
- sid_P^i: the session identifier

[3]When CPace is run inside of AuCPace [86], the CPace password is output from an earlier phase of AuCPace.

- pid_P^i: the partner identifier (who U believes they are communicating with)
- sk_P^i: the session key

Adversarial interaction. The adversary in the security model has full control over the network. The adversary initiates all actions, controls delivery of all protocol messages, and can create, modify, delay, repeat, or delete messages. The adversary interacts with honest participants via calls to the following oracles. In square brackets at the end of each query's description is the symbol we use to denote the number of queries made to that oracle.

- $\mathsf{Send}(P, i, M)$: Captures an active attack. An adversary-selected message M is sent to instance π_P^i, which processes it based on its current state and returns any response message to the adversary. The first call to Send for each instance may include additional context information in M, such as the identity of the intended peer. [Number of Send queries: q_S]
- $\mathsf{Execute}(C, i, S, j)$: Captures the adversary's ability to passively observe honest sessions. If both π_C^i and π_S^j have not yet been used, this query executes the protocol between those two instances. The adversary is provided a transcript of the messages sent by each party. [q_E]
- $\mathsf{Reveal}(P, i)$: The session key sk_P^i is revealed, if it has been set. [q_R]
- $\mathsf{Corrupt}(C, S)$: Reveals $pw_{C,S}$ to the adversary. [q_{Co}]
- $\mathsf{Test}(P, i)$: Issues the challenge for the adversary. Uniformly samples a bit b. If $b = 1$ the session key sk_P^i is revealed to the adversary. If $b = 0$ a uniformly random session key is drawn and returned. This query can be called only once.
- $\cdot(A, B)$: The group operation oracle. [q_G]
- $\mathsf{DLOG}(A, B)$: The discrete logarithm oracle. [q_D]
- Random oracles H_1, H_2. [q_{H_1}, q_{H_2}]

The adversary's goal in the security experiment is, for a sufficiently uncompromised target session, to distinguish the real session key from a random one. At the end of the experiment, the adversary outputs a bit, which is its guess as to whether it was given the real session key or a random one.

Partnering and freshness. Since the adversary can compromise some values and impersonate users, we have to restrict which sessions count as a win for the adversary.

Let π_C^i be a client instance and π_S^j be a server instance with $\mathsf{acc}_C^i = \mathsf{acc}_S^j = \mathsf{true}$. We say that π_C^i and π_S^j are *partnered* if $\mathsf{pid}_C^i = S$, $\mathsf{pid}_S^j = C$, $sk_C^i = sk_S^j$, $\mathsf{sid}_C^i = \mathsf{sid}_S^j$, and there is no other accepting instance with the same sid.

Further, an instance π_P^i is considered *fresh* if all of the following conditions are satisfied:

- a $\mathsf{Reveal}(P, i)$ query has not been made;
- if a partnered instance $\pi_{P'}^j$ exists, then a $\mathsf{Reveal}(P', j)$ query has not been made;
- no $\mathsf{Corrupt}(C, S)$ query has occurred, where C and S are the client and server among P and pid_P^i.

Since we are aiming for security in the quantum-annoying model, we cannot hope to achieve forward secrecy as noted in Section 6.3.1: in CPace$_{\text{base}}$, an adversary with a discrete logarithm oracle could $\mathsf{Execute}$ a session, Test its session key, then $\mathsf{Corrupt}$ the password, hash it to get the corresponding generator, then use its discrete logarithm oracle to find one party's ephemeral shared secret and compute the session key. Thus, our freshness condition above (specifically the third bullet point) does not capture forward secrecy.

Advantage. For a PAKE protocol Π, we say that the adversary *succeeds* if they make a single Test query to an instance that has accepted and terminated and remains fresh throughout the game, and the adversary returns a bit b' that is equal to the bit b that was sampled in the process of answering the Test query. The advantage of the adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}_{\mathrm{BPR}'}^{\Pi}(\mathcal{A}) = 2\Pr[\mathcal{A} \text{ succeeds}] - 1.$$

6.6.2 Security of CPace$_{\text{base}}$

The CPace$_{\text{core}}$ problem defined in Section 6.4 is somewhat unnatural and rather complex, but the benefit of that complexity is that it captures in a single problem all of the characteristics needed to prove the security of CPace$_{\text{base}}$ in the BPR$'$ model.

In CPace$_{\text{base}}$, session identifiers are externally provided. For example, when CPace$_{\text{base}}$ is run as a sub-protocol of AuCPace [86], an earlier stage of AuCPace establishes the a session identifier. For the purposes of the proof, we will assume that session identifiers are provided by the adversary to sessions, with the constraint that, for any session identifier, the adversary may initiate at most one honest client session and at most one honest server session with that session identifier. (This corresponds to the idea that each honest party contributes something fresh and unique-within-that-party to the identifier of each session they participate in, which is the case with how session identifiers are established in AuCPace.)

Theorem 18. *Let $\mathcal{G}$ be a cyclic group of prime order p, and let $H_1 : \{0,1\}^* \to \mathcal{G}$ and $H_2 : \{0,1\}^* \to \{0,1\}^\lambda$ be random oracles. Let $\mathcal{P}$ be a password space of size N. If the* $\text{CPace}_{\text{core}}$ *problem is hard in the generic group model (with a discrete logarithm oracle) for $\mathcal{G}$, then the* $\text{CPace}_{\text{base}}$ *protocol is secure. In particular, if $\mathcal{A}$ is an adversary against* $\text{CPace}_{\text{base}}$ *in the* BPR' *model, then there exists an adversary $\mathcal{B}$ against* $\text{CPace}_{\text{core}}$ *such that*

$$\mathbf{Adv}^{\text{BPR}'}_{\text{CPace}_{\text{base}}}(\mathcal{A}) \leq \frac{4q_{H_2}}{p} + \mathbf{Adv}^{\text{CPace}_{\text{core}}}(\mathcal{B}) \ ,$$

where $\mathcal{A}$ makes at most q_{H_2} queries to H_2. Moreover, the running time of $\mathcal{B}$ is about the same as that of $\mathcal{A}$, and the number of queries $\mathcal{B}$ makes to its $\text{CPace}_{\text{core}}$ *oracles, in terms of the number of queries $\mathcal{A}$ makes to its* $\text{CPace}_{\text{base}}$ *oracles, is as follows:*

- $\cdot$ *(the group operation oracle):* $q^{\mathcal{B}}_{\mathcal{G}} = q^{\mathcal{A}}_{\mathcal{G}}$
- DLOG*:* $q^{\mathcal{B}}_{D} = q^{\mathcal{A}}_{D}$
- *H:* $q^{\mathcal{B}}_{H} \leq q^{\mathcal{A}}_{H_2}$
- GetGen*:* $q^{\mathcal{B}}_{G} \leq q^{\mathcal{A}}_{H_1}$
- NewInstance*:* $q^{\mathcal{B}}_{N} \leq q^{\mathcal{A}}_{E} + q^{\mathcal{A}}_{S} + q^{\mathcal{A}}_{Co} + q^{\mathcal{A}}_{H_1}$
- Challenge *queries of type 1:*[4] $q^{\mathcal{B}}_{C1} \leq q^{\mathcal{A}}_{E}$
- Challenge *queries of type 2:* $q^{\mathcal{B}}_{C2} \leq q^{\mathcal{A}}_{S}$
- GetTarget*:* $q^{\mathcal{B}}_{T} \leq q^{\mathcal{A}}_{Co}$.

Combining Theorem 18 with Theorem 17 yields:

[4]We distinguish Challenge queries that do submit either $(i, 0, V_i)$ or $(i, 1, U_i)$ and Challenge queries that do not submit either of those as type 1 and type 2, respectively. This is because the bounds in Theorem 17 about $\text{CPace}_{\text{core}}$ only care about type 2 Challenge queries.

Corollary 8. *In the generic group model (with a discrete logarithm oracle) for a group $\mathcal{G}$ of order p, for any adversary $\mathcal{A}$ making q_{H_1} H_1 and q_{H_2} H_2 random oracle queries, q_S* Send *queries, $q_{\mathcal{G}}$ group operation queries, and q_D discrete logarithm queries, the advantage of $\mathcal{A}$ in breaking the security of* $\text{CPace}_{\text{base}}$ *with a password dictionary of size N is at most*

$$\mathbf{Adv}_{\text{BPR}'}^{\text{CPace}_{\text{base}}}(\mathcal{A}) \leq \frac{q_D + q_S}{N} + \frac{4q_{H_2} + O(q_{H_1}^2 + q_D q_{\mathcal{G}}^2)}{p} \ .$$

of Theorem 18. We give a reduction $\mathcal{B}$ that, using a $\text{CPace}_{\text{core}}$ challenger, simulates the BPR' security experiment for $\text{CPace}_{\text{base}}$ to $\mathcal{A}$.

The idea behind the simulation $\mathcal{B}$ is as follows. $\mathcal{B}$ maintains a mapping ctr of how $\text{CPace}_{\text{base}}$ user pairs (C, S) and matching sessions (C, S, sid) map on to $\text{CPace}_{\text{core}}$ instances. Recall that calling $\text{CPace}_{\text{core}}.\mathsf{NewInstance}$ with a previously used counter will cause the $\text{CPace}_{\text{core}}$ instance to re-use the same target index; this will correspond to sessions between the same pair of users using the same password; and calling $\text{CPace}_{\text{core}}.\mathsf{GetTarget}$ will allow $\mathcal{B}$ to answer password $\mathsf{Corrupt}$ queries. $\mathcal{B}$ will use the $\text{CPace}_{\text{core}}.\mathsf{NewInstance}$ oracle to simulate message generation and the $\text{CPace}_{\text{core}}.\mathsf{Challenge}$ oracle to compute session keys. One significant difference in $\mathcal{B}$'s simulation is that all session keys – even those returned by Reveal, not just the one returned by Test – are either real or random depending on the hidden secret s of the $\text{CPace}_{\text{core}}$ game. But this will not be a problem, as detecting this in the random oracle model requires a query to the random oracle H_2 which is forwarded to the $\text{CPace}_{\text{core}}.H$ oracle, and would lead to a win in the $\text{CPace}_{\text{core}}$ game.

Initialize $\mathsf{ctr}^* \leftarrow 0$. Define the following subroutine:

- $\text{GETUV}(C, S, \mathsf{sid})$:
    1. If $\mathsf{ctr}_{C,S,\mathsf{sid}}$ is defined: return $(U_{C,S,\mathsf{sid}}, V_{C,S,\mathsf{sid}})$.
    2. Else if $\mathsf{ctr}_{C,S}$ is defined: set $(U_{C,S,\mathsf{sid}}, V_{C,S,\mathsf{sid}}) \leftarrow \text{CPace}_{\text{core}}.\mathsf{NewInstance}(\mathsf{ctr}_{C,S})$, increment ctr^*, and set $\mathsf{ctr}_{C,S,\mathsf{sid}} \leftarrow \mathsf{ctr}^*$. Return $(U_{C,S,\mathsf{sid}}, V_{C,S,\mathsf{sid}})$.
    3. Else: Set $(U_{C,S,\mathsf{sid}}, V_{C,S,\mathsf{sid}}) \leftarrow \text{CPace}_{\text{core}}.\mathsf{NewInstance}(\perp)$, increment ctr^*, and set $\mathsf{ctr}_{C,S}$ and $\mathsf{ctr}_{C,S,\mathsf{sid}}$ to ctr^*.

$\mathcal{B}$ answers queries from $\mathcal{A}$ as follows:

- $\mathsf{Execute}(C, i, S, j, \mathsf{sid})$: We use the $\mathsf{NewInstance}$ (via GETUV) and $\mathsf{Challenge}$ oracles of the $\text{CPace}_{\text{core}}$ challenger to generate a transcript and session key, and receive a session identifier from the adversary, which must be previously unused by C and S. Set $\mathsf{sid}_C^i, \mathsf{sid}_S^j \leftarrow \mathsf{sid}$. Set $(U, V) \leftarrow \text{GETUV}(C, S, \mathsf{sid})$. Set $sk_C^i \leftarrow \text{CPace}_{\text{core}}.\mathsf{Challenge}(\mathsf{ctr}_{C,S,\mathsf{sid}}, 0, V)$ and $sk_S^j \leftarrow sk_C^i$. Return transcript (U, V).

- $\mathsf{Send}(C, i, M = (\mathsf{sid}, S))$ to a client C: We use the $\mathsf{NewInstance}$ oracle of the $\mathrm{CPace}_{\mathrm{core}}$ challenger (via $\textsc{getUV}$) to generate the message for the client side of a session. Set $\mathsf{sid}_C^i \leftarrow \mathsf{sid}$. Run $\textsc{getUV}(C, S, \mathsf{sid})$. Return outgoing message U_C^i.

- $\mathsf{Send}(S, j, M = (C, \mathsf{sid}, U))$ to a server S: We use the $\mathsf{NewInstance}$ oracle of the $\mathrm{CPace}_{\mathrm{core}}$ challenger (via $\textsc{getUV}$) to generate the message for the server side of a session, and the $\mathsf{Challenge}$ oracle to generate the session key. Set $\mathsf{sid}_S^j \leftarrow \mathsf{sid}$. Run $\textsc{getUV}(C, S, \mathsf{sid})$ and set $sk_S^j \leftarrow \mathrm{CPace}_{\mathrm{core}}.\mathsf{Challenge}(\mathsf{ctr}_{C,S,\mathsf{sid}}, 1, U)$. Return outgoing message V_S^j.

- $\mathsf{Send}(C, i, M = V)$ to a client C: We use the $\mathsf{Challenge}$ oracle of the $\mathrm{CPace}_{\mathrm{core}}$ challenger to complete the session. Set $sk_C^i \leftarrow \mathrm{CPace}_{\mathrm{core}}.\mathsf{Challenge}(\mathsf{ctr}_{C,S,\mathsf{sid}_C^i}, 0, V)$.

- $\mathsf{Reveal}(P, i)$: Return sk_P^i, if it has been set.

- $\mathsf{Corrupt}(C, S)$: We use the $\mathsf{GetTarget}$ oracle of the $\mathrm{CPace}_{\mathrm{core}}$ challenger to let the $\mathrm{CPace}_{\mathrm{core}}$ challenger pick which password is being used for this client-server pair. If $pw_{C,S}$ is set, return it. If $\mathsf{ctr}_{C,S}$ is not defined, run $\textsc{getUV}(C, S, \mathsf{sid})$ for a random, unused sid. Let $t \leftarrow \mathrm{CPace}_{\mathrm{core}}.\mathsf{GetTarget}(\mathsf{ctr}_{C,S})$. Set $pw_{C,S} \leftarrow \mathcal{P}[t]$ (i.e., the tth password in password dictionary $\mathcal{P}$). Return $pw_{C,S}$.

- $\mathsf{Test}(P, i)$: Return sk_P^i.

- $\cdot(A, B)$ (the group operation oracle): Return $\mathrm{CPace}_{\mathrm{core}}. \cdot (A, B)$.

- $\mathsf{DLOG}(A, B)$: Return $\mathrm{CPace}_{\mathrm{core}}.\mathsf{DLOG}(A, B)$.

- $H_1(\mathsf{sid} \| pw \| \textsc{oc}(C, S))$: Separate out $\textsc{oc}(C, S)$ into C, S (recall that the set of clients and servers is distinct so this is possible). If $\mathsf{ctr}_{C,S,\mathsf{sid}}$ is not defined, then run $\textsc{getUV}(C, S, \mathsf{sid})$. Let t be the index of pw in the password dictionary $\mathcal{P}$. Return $\mathrm{CPace}_{\mathrm{core}}.\mathsf{GetGen}(\mathsf{ctr}_{C,S,\mathsf{sid}}, t)$. We assume that the adversary does not elect to make queries of this form, as they have no bearing on the protocol.

- $H_2(\mathsf{sid} \| K \| \textsc{oc}(U, V))$: If this query has already been asked, answer as before. Otherwise:

  - If there is no C, S such that $\mathsf{ctr}_{C,S,\mathsf{sid}}$ is defined, we maintain the random oracle ourselves using a table $\mathcal{H}$. If $\mathcal{H}[\mathsf{sid} \| K \| \textsc{oc}(U, V)]$ is not defined, select it uniformly at random from the set $\mathrm{CPace}_{\mathrm{core}}.\mathcal{C}$. Return $\mathcal{H}[\mathsf{sid} \| K \| \textsc{oc}(U, V)]$.
  - If there exists C, S such that $\mathsf{ctr}_{C,S,\mathsf{sid}}$ is defined: Return $\mathrm{CPace}_{\mathrm{core}}.H(\mathsf{ctr}_{C,S,\mathsf{sid}}, K, \textsc{oc}(U, V))$.

$\mathcal{B}$ runs $\mathcal{A}$ until either $\mathcal{B}$ wins the $\mathrm{CPace}_{\mathrm{core}}$ game, or $\mathcal{A}$ terminates and outputs a bit, in which case $\mathcal{B}$ uses that bit as its guess of s in the $\mathrm{CPace}_{\mathrm{core}}$ game.

In most ways, $\mathcal{B}$ correctly simulates the execution of $\mathrm{CPace}_{\mathrm{base}}$ to $\mathcal{A}$. The password $pw_{C,S}$ for a client-server pair corresponds to the tth password in the dictionary, where t is the target index of all sessions between C and S (since they use the same $\mathsf{ctr} = \mathsf{ctr}_{C,S}$ in calls to $\mathrm{CPace}_{\mathrm{core}}.\mathsf{NewInstance}$). The messages in the $\mathsf{Execute}$ and Send queries are distributed exactly as in $\mathrm{CPace}_{\mathrm{base}}$. The responses to $\mathsf{Corrupt}$, $\cdot$, DLOG, and H_1 are also all distributed correctly.

Assuming the Test session $\pi_{P^*}^{i^*}$ remains fresh means that the adversary has not made a $\mathsf{Corrupt}(C^*, S^*)$ query for the client C^* and server S^* in the test session prior to the Test query. Therefore, $\mathcal{B}$ has not made a $\mathsf{GetTarget}(\mathsf{ctr}_{C^*,S^*})$ query to $\mathrm{CPace}_{\mathrm{core}}$ prior to the $\mathsf{Challenge}$ query being issued for the instance corresponding to the Test session, and thus the session key in the test session is real-or-random, depending on the secret bit s of the $\mathrm{CPace}_{\mathrm{core}}$ game. Thus the response to the Test query is properly distributed.

The session keys set during $\mathsf{Execute}$ and Send queries are *not* perfectly simulated; in the BPR' experiment for $\mathrm{CPace}_{\mathrm{base}}$, only the response to the Test query should be real-or-random, but in $\mathcal{B}$'s simulation, all session keys are real-or-random (since they all are generated by a call to $\mathrm{CPace}_{\mathrm{core}}.\mathsf{Challenge}$) and thus all responses from Reveal are real-or-random. Additionally, responses to H_2 are not simulated correctly when called with a sid which has not been already passed to a client or server instance via $\mathsf{Execute}$ or Send. The rest of the proof focuses on why these inconsistencies are not a problem.

First we consider whether an adversary can detect that responses from a $\mathsf{Reveal}(P, i)$ query are real-or-random, not real. Since session keys are the output of the random oracle H_2, this would mean that the adversary has to query H_2 on $\mathsf{sid}_P^i \| K \| \mathrm{OC}(U, V)$ where $K = DH(U, V)$ and U, V are the messages used by instance π_P^i; call this E_1. Let C and S be the client and server instance respectively among P and pid_P^i. If the adversary has called $\mathsf{Corrupt}(C, S)$ or $\mathsf{Corrupt}(S, C)$ prior to this Reveal query, then $\mathcal{B}$ will have called $\mathsf{GetTarget}(\mathsf{ctr}_{C,S})$, and $\mathrm{CPace}_{\mathrm{core}}$ is defined such that a subsequent call to $\mathsf{Challenge}$ for any instance linked to $\mathsf{ctr}_{C,S}$ returns real session keys, regardless of the hidden bit. If the adversary has not called $\mathsf{Corrupt}(C, S)$ or $\mathsf{Corrupt}(S, C)$ prior to this reveal query, then $\mathcal{B}$ will not have called $\mathsf{GetTarget}(\mathsf{ctr}_{C,S})$. Since π_P^i is a session at a party simulated by $\mathcal{B}$, either $U = U_{C,S,\mathsf{sid}_P^i}$ (if $P = C$) or $V = V_{C,S,\mathsf{sid}_P^i}$ (if $P = S$). Either way, at least one of U, V was the output of a call to $\mathsf{NewInstance}$ for $\mathsf{ctr}_{C,S,\mathsf{sid}_P^i}$. By construction, $\mathcal{B}$ relays an H_2 query for a defined sid_P^i to $\mathrm{CPace}_{\mathrm{core}}.H$. Hence, $\mathcal{B}$ queries $\mathrm{CPace}_{\mathrm{core}}.H$ with either $(i^*, W_{i^*,0}^{u_{i^*}}, \mathrm{OC}(U_{i^*}, W_{i^*,0}))$ (in the case where $P = C$, taking $i^* = \mathsf{ctr}_{C,S,\mathsf{sid}_P^i}$, $U_{i^*} = U = U_{C,S,\mathsf{sid}_P^i}$, $W_{i^*,0} = V$) or $(i^*, W_{i^*,1}^{v_{i^*}}, \mathrm{OC}(V_{i^*}, W_{i^*,1}))$ (in the case where $P = S$, taking $i^* = \mathsf{ctr}_{C,S,\mathsf{sid}_P^i}$, $V_{i^*} = V = V_{C,S,\mathsf{sid}_P^i}$, $W_{i^*,1} = U$), both of which are immediately winning queries to $\text{CPace}_{\text{core}}$. In other words, $\Pr[\mathsf{E}_1] \leq \epsilon$, where ϵ is the probability of winning $\text{CPace}_{\text{core}}$ with the number of queries made by $\mathcal{B}$.

Now we consider whether an adversary can detect that responses to H_2 are not simulated correctly when called with a sid which has not been already passed to a client or server instance via Execute or Send; call this E_2. We permitted the adversary to choose sids for sessions, so we will not assume that sids are unpredictable before used in an session at an honest party. However, when a session is activated at an honest party using either Execute or Send, a fresh U or V value (depending on whether it is a client or server session) will be chosen by the simulator. The chance that a U or V value used in an H_2 query for an undefined sid is the same as one of the U or V values chosen when an honest party runs on that sid is at most $4/p$, where p is the order of the group. If $\mathcal{A}$ makes q_{H_2} H_2 queries, then the probability the simulation is invalid is at most $\Pr[\mathsf{E}_2] \leq 4q_{H_2}/p$.

Note that when a Corrupt query is made, the induced GetTarget query in $\text{CPace}_{\text{core}}$ causes the oracle H to be reprogrammed, which in turn reprograms H_2. Thus we must consider the adversary's ability to notice such a reprogramming. However the only way to notice a reprogramming is to already have queried H_2 on a point that will induce a query to H on a reprogrammed point. The reprogrammed points are those which will eventually be used to induce session keys, queries of the form $H(\mathsf{ctr}, W_{i,0}^{u_i}, \text{OC}(U_i, W_{i,0}))$ or $H(\mathsf{ctr}, W_{i,1}^{v_i}, \text{OC}(V_i, W_{i,1}))$. But these are precisely the points that, if queried to H before a Corrupt query, then the reduction $\mathcal{B}$ wins the $\text{CPace}_{\text{core}}$ game. As a result, any advantage the adversary has in noticing reprogrammed points is exactly conferred to an advantage $\mathcal{B}$ has in winning $\text{CPace}_{\text{core}}$, and we need not be concerned with the probability this happens.

Assuming neither E_1 nor E_2 occur, there is no inconsistency in $\mathcal{B}$'s simulation of the BPR$'$ security game to $\mathcal{A}$. If $\mathcal{A}$'s output changes based on whether the answer to the Test query was real or random, then $\mathcal{B}$'s output will change based on whether the secret s in $\text{CPace}_{\text{core}}$ is 0 or 1. Combining these two statements yields

$$\mathbf{Adv}^{\text{BPR}'}_{\text{CPace}_{\text{base}}}(\mathcal{A}) \leq \frac{4q_{H_2}}{p} + \mathbf{Adv}^{\text{CPace}_{\text{core}}}(\mathcal{B}) \ .$$

The runtime of $\mathcal{B}$ is the same as the runtime of $\mathcal{A}$, plus a small bookkeeping overhead for each query. Moreover, $\mathcal{B}$ makes a number of queries to oracles in correspondence with the statement of Theorem 18. □

Chapter 7

Conclusion

In this thesis, we have addressed a series of open questions in the study of post-quantum cryptography. We have shown the flexibility of several post-quantum signature and key establishment schemes, by establishing how they can be adapted to support features that have been designed with the structure of the discrete logarithm problem.

In Chapter 2 we considered the primitive of key-blinding and how it could be made post-quantum. We show how it could be constructed with four different post-quantum signature schemes. These used different methods to accomplish the task of key-blinding. CSI-FiSh was amenable to key-blinding because it instantiates a group action. DILITHIUM-QROM was similarly amenable, because of the additive homomorphic structure between the secret and public key space. Picnic and LegRoast supported key-blinding because they both are built out of generic zero-knowledge proof systems, which can be adapted to the needs of key-blinding. We also presented a generic proof framework for thinking about the unlinkability property of key-blinding schemes and put it into action with each of the schemes presented.

We also have considered the primitive of Updatable Public Key Encryption in Chapter 3, and similarly shown how it can be instantiated with the key-exchange scheme CSIDH. More specifically we borrowed techniques from the signature scheme CSI-FiSh, which provided enough mathematical structure to instantiate a provably secure UPKE scheme.

As well as establishing how to extend post-quantum schemes, we have shown several results that inform how we can model and reason about quantum adversaries. In Chapter 4, we investigated and proved several results about the ability of quantum adversaries to find collisions in non-uniform functions. We were able to characterize (usually tightly) the number of queries necessary and sufficient in many configurations.

In Chapter 5 we established a fundamental result on the instantiability of the quantum random oracle model: that there exist signature scheme secure in the QROM that are insecure when the random oracle is replaced with any real-world hash function. This extends existing, similar results on the ROM [46] that have been highly influential in changing how researchers view these proof models.

Finally, in Chapter 6, we considered a limitation of quantum computation. For an adversary restricted to using quantum computation to solve discrete logarithms, we showed certain password-authenticated key exchange protocols can remain somewhat secure, requiring many discrete logarithms to be solved.

7.1 Future Work

While this thesis has solved many problems in the development of post-quantum cryptography, many more remain. More complicated protocols, such as PAKEs [99], zero-knowledge proof systems [20], group signatures [36, 131], threshold schemes [56, 106] and more are seeing increased usage, especially as secure messaging protocols like Signal and advanced usages of cryptography in blockchains are becoming more commonplace. Work on the development of suitable post-quantum replacements for these is still early, and standardization is even further off.

In addition to constructing primitives, work on quantum adversarial models in ongoing, especially developing proof techniques in such models. There have been tremendous new techniques and results in recent years [12, 54, 64, 107]. However many of these techniques are only comprehensible to those who have expertise in quantum information theory, and making them more accessible to cryptographers without such expertise is important ongoing work.

The concept of quantum annoyingness also leaves many potential future research directions. The first is in the model itself. The model we used for quantum annoyingness is restrictive, as it only allows the usage of quantum computers for solving discrete logarithms. There is a great need to either *validate* this model, and provide better evidence that solving individual discrete logarithms is all an adversary would want to do, or *expand* the model, showing the security of the protocol under more realistic assumptions of quantum computation. Quantum annoyingness can also be shown for other schemes. We established it for the PAKE protocol CPace, but there are other PAKE protocols that are plausibly quantum annoying. It is an open question whether other schemes, such as continuous key agreement protocols, can be argued to be quantum annoying.

Post-quantum cryptography is in somewhat of a race against the physicists and engineers working on developing scalable quantum computers. With proper research and development, we can be prepared for a post-quantum world, and enjoy the benefits of quantum computation without the general public ever having noticed the security disaster that was neatly avoided.

References

[1] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM (JACM)*, 51(4):595–605, 2004.

[2] Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi. Tightly secure signatures from lossy identification schemes. *Journal of Cryptology*, 29(3):597–631, July 2016.

[3] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of cpace. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 711–741. Springer Heidelberg, 2021.

[4] National Security Agency. Cryptography today, 2015. Archived on 23 November 2015, tinyurl.com/SuiteB.

[5] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and Jurjen Bos. HQC. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[6] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 411–439. Springer, Heidelberg, December 2020.

[7] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson,

and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[8] Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. Deterministic wallets in a quantum world. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1017–1031. ACM Press, November 2020.

[9] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 248–277. Springer, Heidelberg, August 2020.

[10] Andris Ambainis. Quantum walk algorithm for element distinctness. In *45th Annual Symposium on Foundations of Computer Science*, pages 22–31. IEEE Computer Society Press, October 2004.

[11] Andris Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory of Computing*, 1(3):37–46, 2005.

[12] Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 269–295. Springer, Heidelberg, August 2019.

[13] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. Quantum attacks on classical proof systems: The hardness of quantum rewinding. In *55th Annual Symposium on Foundations of Computer Science*, pages 474–483. IEEE Computer Society Press, October 2014.

[14] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, and Santosh Ghosh. BIKE. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[15] Fatih Balli, Paul Rösler, and Serge Vaudenay. Determining the core primitive for optimally secure ratcheting. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 621–650. Springer, Heidelberg, December 2020.

[16] Marko Balogh, Edward Eaton, and Fang Song. Quantum collision-finding in non-uniform random functions. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 467–486. Springer, Heidelberg, 2018.

[17] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Message Layer Security (MLS) Protocol. https://datatracker.ietf.org/doc/pdf/draft-ietf-mls-protocol-12, October 2021.

[18] Paulo S. L. M. Barreto, Jefferson E. Ricardini, Marcos A. Simplicio Jr., and Harsh Kupwade Patil. qSCMS: Post-quantum certificate provisioning process for V2X. Cryptology ePrint Archive, Report 2018/1247, 2018. https://eprint.iacr.org/2018/1247.

[19] Carsten Baum, Cyprien de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 266–297. Springer, Heidelberg, May 2021.

[20] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, Heidelberg, May 2020.

[21] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188. Springer, Heidelberg, May 2004.

[22] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

[23] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, Heidelberg, May 2000.

[24] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, November 1993.

[25] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, Heidelberg, May 1995.

[26] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, Heidelberg, May 1996.

[27] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, Heidelberg, May / June 2006.

[28] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[29] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009: 12th International Conference on Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 33–48. Springer, Heidelberg, September 2009.

[30] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142. Springer, Heidelberg, September / October 2011.

[31] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2129–2146. ACM Press, November 2019.

[32] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak sponge function family, 2007. http://keccak.noekeon.org/.

[33] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. Cryptanalysis of the Legendre PRF and generalizations. *IACR Transactions on Symmetric Cryptology*, 2020(1):313–330, 2020.

[34] Ward Beullens and Cyprien de Saint Guilhem. LegRoast: Efficient post-quantum signatures from the Legendre PRF. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 130–150. Springer, Heidelberg, 2020.

[35] Ward Beullens and Cyprien Delpech de Saint Guilhem. Legroast. GitHub Repository. https://github.com/WardBeullens/LegRoast, 2020. Accessed May 2022.

[36] Ward Beullens, Samuel Dobson, Shuichi Katsumata, Yi-Fu Lai, and Federico Pintore. Group signatures and more from isogenies and lattices: Generic, simple, and efficient. To appear in the proceedings of Eurocrypt 2022, 2022. https://ia.cr/2021/1366.

[37] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh. GitHub Repository. https://github.com/KULeuven-COSIC/CSI-FiSh, 2019. Accessed May 2021.

[38] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, Heidelberg, December 2019.

[39] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris, May 2018.

[40] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun

Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69. Springer, Heidelberg, December 2011.

[41] Dan Boneh, Saba Eskandarian, Sam Kim, and Maurice Shih. Improving speed and security in updatable encryption schemes. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 559–589. Springer, Heidelberg, December 2020.

[42] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

[43] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, jun 1998.

[44] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum algorithm for the collision problem. In *Encyclopedia of Algorithms*, pages 1662–1664. 2016.

[45] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for Signal's X3DH handshake. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, volume 12804 of *Lecture Notes in Computer Science*, pages 404–430. Springer, Heidelberg, 2020.

[46] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218. ACM Press, May 1998.

[47] Ran Canetti, Oded Goldreich, and Shai Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 40–57. Springer, Heidelberg, February 2004.

[48] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. GeMSS. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[49] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and

Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, Heidelberg, December 2018.

[50] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1825–1842. ACM Press, October / November 2017.

[51] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. *Journal of Cryptographic Engineering*, 2021.

[52] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[53] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 1–29. Springer, Heidelberg, December 2019.

[54] Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. On the compressed-oracle technique, and post-quantum security of proofs of sequential work. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 598–629. Springer, Heidelberg, October 2021.

[55] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, Heidelberg, August 2000.

[56] Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 169–186. Springer, Heidelberg, 2020.

[57] Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, Christian Schaffner, and Dominique Unruh. Post-quantum security of the sponge construction. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 185–204. Springer, Heidelberg, 2018.

[58] Ivan Damgård. On the randomness of Legendre and Jacobi sequences. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 163–172. Springer, Heidelberg, August 1990.

[59] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[60] Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789. Springer, Heidelberg, May 2019.

[61] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93. Springer, Heidelberg, December 2020.

[62] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias Kannwischer, and Jacques Patarin. Rainbow. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[63] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 254–285. Springer, Heidelberg, November 2021.

[64] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019,*

Part II, volume 11693 of *Lecture Notes in Computer Science*, pages 356–383. Springer, Heidelberg, August 2019.

[65] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839.

[66] Edward Eaton. Leighton-Micali hash-based signatures in the quantum random-oracle model. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017: 24th Annual International Workshop on Selected Areas in Cryptography*, volume 10719 of *Lecture Notes in Computer Science*, pages 263–280. Springer, Heidelberg, August 2017.

[67] Edward Eaton, David Jao, Chelsea Komlo, and Youcef Mokrani. Towards post-quantum key-updatable public-key encryption via supersingular isogenies. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021: 28th Annual International Workshop on Selected Areas in Cryptography*, volume 13203 of *Lecture Notes in Computer Science*, pages 461–482. Springer, Heidelberg, 2021.

[68] Edward Eaton and Fang Song. Making existential-unforgeable signatures strongly unforgeable in the quantum random-oracle model. In *10th Conference on the Theory of Quantum Computation, Communication and Cryptography – TQC 2015*, volume 44 of *LIPIcs*, pages 147–162. Schloss Dagstuhl, 2015.

[69] Edward Eaton and Fang Song. A note on the instantiability of the quantum random oracle. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 503–523. Springer, Heidelberg, 2020.

[70] Edward Eaton and Douglas Stebila. The "quantum annoying" property of password-authenticated key exchange protocols. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pages 154–173. Springer, Heidelberg, 2021.

[71] Edward Eaton, Douglas Stebila, and Roy Stracovsky. Post-quantum key-blinding for authentication in anonymity networks. In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology - LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America*, volume 12912 of *Lecture*

Notes in Computer Science, pages 67–87, Bogotá, Colombia, October 2021. Springer, Heidelberg.

[72] Ehsan Ebrahimi and Dominique Unruh. Quantum collision-resistance of non-uniformly distributed functions: upper and lower bounds. *Quantum Inf. Comput.*, 18(15&16):1332–1349, 2018.

[73] Federal Office for Information Security (BSI). Advanced security mechanism for machine readable travel documents (extended access control (EAC), password authenticated connection establishment (PACE), and restricted identification (RI)), 2008. SI-TR-03110, Version 2.0, https://www.bsi.bund.de/EN/Service-Navi/Publications/TechnicalGuidelines/TR03110/BSITR03110.html.

[74] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Heidelberg, August 1987.

[75] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013.

[76] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, May / June 2009.

[77] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206. ACM Press, May 2008.

[78] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, Heidelberg, August 2013.

[79] Vlad Gheorghiu and Michele Mosca. Benchmarking the quantum cryptanalysis of symmetric, public-key and hash-based cryptographic schemes, 2019. https://arxiv.org/abs/1902.02332.

[80] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 1069–1083. USENIX Association, August 2016.

[81] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Annual Symposium on Foundations of Computer Science*, pages 102–115. IEEE Computer Society Press, October 2003.

[82] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 430–443. ACM Press, October 2016.

[83] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM Press, May 1996.

[84] Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015: 19th International Conference on Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 497–504. Springer, Heidelberg, January 2015.

[85] Björn Haase. CPace, a balanced composable PAKE. Internet-Draft draft-haase-cpace-01, February 2020. `http://www.ietf.org/internet-drafts/draft-haase-cpace-01.txt`.

[86] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):1–48, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/7384`.

[87] Sean Hallgren, Adam Smith, and Fang Song. Classical cryptographic protocols in a quantum world. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 411–428. Springer, Heidelberg, August 2011.

[88] Eduard Hauck, Eike Kiltz, Julian Loss, and Ngoc Khanh Nguyen. Lattice-based blind signatures, revisited. In Daniele Micciancio and Thomas Ristenpart, editors,

Advances in Cryptology – CRYPTO 2020, Part II, volume 12171 of *Lecture Notes in Computer Science*, pages 500–529. Springer, Heidelberg, August 2020.

[89] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, Heidelberg, November 2017.

[90] Nicholas Hopper. Proving security of Tor's hidden service identity blinding protocol. https://www-users.cs.umn.edu/~hoppernj/basic-proof.pdf, 2013.

[91] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[92] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 387–416. Springer, Heidelberg, March 2016.

[93] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 21–30. ACM Press, June 2007.

[94] David P. Jablon. Strong password-only authenticated key exchange. *Comput. Commun. Rev.*, 26(5):5–26, 1996.

[95] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 33–62. Springer, Heidelberg, August 2018.

[96] Rahul Jain, Alexandra Kolla, Gatis Midrijanis, and Ben W Reichardt. On parallel composition of zero-knowledge proofs with black-box quantum simulators. *Quan-*

tum Information & Computation, 9(5):513–532, 2009. Available at arXiv:quant-ph/0607211.

[97] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

[98] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34. Springer, Heidelberg, November / December 2011.

[99] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 456–486. Springer, Heidelberg, April / May 2018.

[100] Yao Jiang. The direction of updatable encryption does not matter much. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 529–558. Springer, Heidelberg, December 2020.

[101] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 159–188. Springer, Heidelberg, May 2019.

[102] Dmitry Khovratovich. Key recovery attacks on the Legendre PRFs within the birthday bound. Cryptology ePrint Archive, Report 2019/862, 2019. `https://eprint.iacr.org/2019/862`.

[103] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 552–586. Springer, Heidelberg, April / May 2018.

[104] Eike Kiltz, Adam O'Neill, and Adam D. Smith. Instantiability of RSA-OAEP under chosen-plaintext attack. *Journal of Cryptology*, 30(3):889–919, July 2017.

[105] Neal Koblitz and Alfred J. Menezes. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptogr.*, 77(2-3):587–610, 2015.

[106] Chelsea Komlo and Ian Goldberg. FROST: flexible round-optimized schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65. Springer, Heidelberg, 2020.

[107] Veronika Kuchta, Amin Sakzad, Damien Stehlé, Ron Steinfeld, and Shifeng Sun. Measure-rewind-measure: Tighter quantum random oracle model proofs for one-way to hiding and CCA security. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 703–728. Springer, Heidelberg, May 2020.

[108] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes, and Cryptography*, 75(3):565–599, 2015.

[109] Zhen Liu, Khoa Nguyen, Guomin Yang, Huaxiong Wang, and Duncan S. Wong. A lattice-based linkable ring signature supporting stealth addresses. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019: 24th European Symposium on Research in Computer Security, Part I*, volume 11735 of *Lecture Notes in Computer Science*, pages 726–746. Springer, Heidelberg, September 2019.

[110] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[111] Moxie Marlinspike and Trevor Perrin. The Double Ratchet Algorithm, 2016. https://signal.org/docs/specifications/doubleratchet/.

[112] Moxie Marlinspike and Trevor Perrin. The X3DH Key Agreement Protocol, 2016. https://signal.org/docs/specifications/x3dh/.

[113] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951

of *Lecture Notes in Computer Science*, pages 21–39. Springer, Heidelberg, February 2004.

[114] Ueli M. Maurer and Stefan Wolf. Lower bounds on generic algorithms in groups. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 72–84. Springer, Heidelberg, May / June 1998.

[115] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[116] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[117] Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. On the security of the schnorr signature scheme and DSA against related-key attacks. In Soonhak Kwon and Aaram Yun, editors, *ICISC 15: 18th International Conference on Information Security and Cryptology*, volume 9558 of *Lecture Notes in Computer Science*, pages 20–35. Springer, Heidelberg, November 2016.

[118] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

[119] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, Heidelberg, August 2002.

[120] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions, 2014. Available at http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf.

[121] National Institute of Standards and Technology. Post quantum cryptography call for proposals, 2017. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals. Accessed April 2021.

[122] National Institute of Standards and Technology. FIPS 180-1: Secure hash standard, April 1995.

[123] Chris Peikert. He gives C-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492. Springer, Heidelberg, May 2020.

[124] Albrecht Petzoldt, Alan Szepieniec, and Mohamed Saied Emam Mohamed. A practical multivariate blind signature scheme. In Aggelos Kiayias, editor, *FC 2017: 21st International Conference on Financial Cryptography and Data Security*, volume 10322 of *Lecture Notes in Computer Science*, pages 437–454. Springer, Heidelberg, April 2017.

[125] Bertram Poettering and Paul Rösler. Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296, 2018. https://eprint.iacr.org/2018/296.

[126] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 3–32. Springer, Heidelberg, August 2018.

[127] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[128] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM Press, May 2005.

[129] Ronald L. Rivest. RFC 1321: The MD5 message-digest algorithm, April 1992. https://www.ietf.org/rfc/rfc1321.txt.

[130] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[131] Masoumeh Shafieinejad and Navid Nasr Esfahani. A scalable post-quantum hash-based group signature. *Des. Codes Cryptogr.*, 89(5):1061–1090, 2021.

[132] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[133] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, Heidelberg, May 1997.

[134] Stanislav Smyshlyaev, Nick Sullivan, and Alexey Melnikov. [Cfrg] Results of the PAKE selection process. CFRG Mailing List, March 2020. https://mailarchive.ietf.org/arch/msg/cfrg/LKbwodpa5yXo6VuNDU66vt_Aca8/.

[135] Nick Sullivan, Stanislav Smyshlyaev, Kenny Paterson, and Alexey Melnikov. Proposed PAKE selection process. CFRG Mailing List, May 2019. https://mailarchive.ietf.org/arch/msg/cfrg/-J43ZsPw2J5MBC-k8y6--kJJtZk/.

[136] Ehsan Ebrahimi Targhi, Gelo Noel Tabia, and Dominique Unruh. Quantum collision-resistance of non-uniformly distributed functions. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 79–85. Springer, Heidelberg, 2016.

[137] Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 192–216. Springer, Heidelberg, October / November 2016.

[138] The Tor Project, Inc. Tor Metrics. https://metrics.torproject.org/, 2020. Accessed September 2021.

[139] The Tor Project, Inc. Tor Rendezvous Specification - Version 3. https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt, 2020.

[140] Steve Thomas. Re: [Cfrg] proposed PAKE selection process. CFRG Mailing list, June 2019. https://mailarchive.ietf.org/arch/msg/cfrg/dtf91cmavpzT47U3AVxrVGNB5UM/.

[141] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 755–784. Springer, Heidelberg, April 2015.

[142] Dominique Unruh. Revocable quantum timed-release encryption. *J. ACM*, 62(6):49:1–49:76, 2015.

[143] Dominique Unruh. Computationally binding quantum commitments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 497–527. Springer, Heidelberg, May 2016.

[144] Loup Vaillant-David. Re: [Cfrg] CPACE: what the "session id" is for? CFRG Mailing List, June 2020. https://mailarchive.ietf.org/arch/msg/cfrg/1b3H_VKyZR-UbE6FoNeOdOOPqRE/.

[145] Michael J. Wiener. Bounds on birthday attack times. Cryptology ePrint Archive, Report 2005/318, 2005. https://eprint.iacr.org/2005/318.

[146] Henry Yuen. A quantum lower bound for distinguishing random functions from random permutations. *Quantum Information & Computation*, 14(13-14):1089–1097, 2014.

[147] Aaram Yun. Generic hardness of the multiple discrete logarithm problem. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 817–836. Springer, Heidelberg, April 2015.

[148] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. Picnic. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[149] Mark Zhandry. How to construct quantum random functions. In *53rd Annual Symposium on Foundations of Computer Science*, pages 679–687. IEEE Computer Society Press, October 2012.

[150] Mark Zhandry. A note on the quantum collision and set equality problems. *Quantum Information & Computation*, 15(7-8):557–567, 2015.

[151] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 239–268. Springer, Heidelberg, August 2019.

[152] Mark Zhandry. Augmented random oracles. In *To Appear in Advances in Cryptology - CRYPTO 2022*. Springer Heidelberg, 2022.